Master's thesis
Master's Programme in Data Science

Network Specialization to Explain the Performance of Sparse Neural Networks

Vili Hätönen

August 13, 2020

Supervisor(s): Dr. Antti Ukkonen, Dr. Joel Pyykkö

Examiner(s): Professor Teemu Roos
Dr. Antti Ukkonen

University of Helsinki
Faculty of Science

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Degree programme
Faculty of Science	Master's Programme in Data Science

Tekijä — Författare — Author

Vili Hätönen

Työn nimi — Arbetets titel — Title

Network Specialization to Explain the Performance of Sparse Neural Networks

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidantal — Number of pages
Master's thesis	August 13, 2020	114

Tiivistelmä — Referat — Abstract

Recently it has been shown that sparse neural networks perform better than dense networks with similar number of parameters. In addition, large overparameterized networks have been shown to contain sparse networks which, while trained in isolation, reach or exceed the performance of the large model. However, the methods to explain the success of sparse networks are still lacking. In this work I study the performance of sparse networks using network's activation regions and patterns, concepts from the neural network expressivity literature.

I define *network specialization*, a novel concept that considers how distinctly a feed forward neural network (FFNN) has learned to processes high level features in the data. I propose Minimal Blanket Hypervolume (MBH) algorithm to measure the specialization of a FFNN. It finds parts of the input space that the network associates with some user-defined high level feature, and compares their hypervolume to the hypervolume of the input space. My hypothesis is that sparse networks specialize more to high level features than dense networks with the same number of hidden network parameters.

Network specialization and MBH also contribute to the interpretability of deep neural networks (DNNs). The capability to learn representations on several levels of abstraction is at the core of deep learning, and MBH enables numerical evaluation of how specialized a FFNN is w.r.t. any abstract concept (a high level feature) that can be embodied in an input. MBH can be applied to FFNNs in any problem domain, e.g. visual object recognition, natural language processing, or speech recognition. It also enables comparison between FFNNs with different architectures, since the metric is calculated in the common input space.

I test different pruning and initialization scenarios on the MNIST Digits and Fashion datasets. I find that sparse networks approximate more complex functions, exploit redundancy in the data, and specialize to high level features better than dense, fully parameterized networks with the same number of hidden network parameters.

ACM Computing Classification System (CCS):
Computing methodologies → Machine learning → Neural networks

Avainsanat — Nyckelord — Keywords

Sparsity, Activation Region, Lottery Ticket Hypothesis, Deep Neural Networks, Bent Hyperplanes

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Contents

1. Introduction

Deep neural networks (DNNs) have become the state of the art models in many domains, including visual object and speech recognition, natural language processing (NLP), genomics, and others [48]. Deep learning is about learning representations of the data on multiple levels of abstraction [4, 24, 48]. An artificial neural network (ANN), which has many hidden layers instead of one, can learn multiple levels of representation better than shallow ANNs with only one hidden layer [4, 65].

It is typical for the contemporary DNNs to have tens of millions of parameters[1]. Models of this scale require years of GPU-time to train [42], and are therefore expensive to train and use. Not only in terms of money and time, but also for the environment [72]: training a large NLP pipeline with tuning and experimenting is estimated to produce the same carbon footprint as the lives of seven average humans[2] [76]. Other drivers for smaller and more efficient models are the memory and computation requirements in restricted environments, like robotics, augmented reality, mobile, and embedded applications [37].

There is a large body of work addressing the prohibitive storage, memory, and computation requirements of DNNs. In model compression [6] and knowledge distillation [33] a smaller model is trained to approximate a larger, better performing one. In the context of neural networks this is called network compression [51], and during the past decade there has been many proposals how it could be implemented[3]. Two popular approaches are network quantization [23] and pruning [50], which also can be used together [28].

In network quantization the storage size of the network's parameters is reduced [23, 40] by changing parameters' datatype from more precise representation (e.g. 64 bit float) to more coarse representation (e.g. 4 bit integer [23], or even binary values [10]). In network pruning some parameters are removed, making the model sparse [29, 91].

[1] From the image domain: AlexNet (2012) $60{\cdot}10^6$ parameters [46], Inception-ResNet-V2 (2016) $56{\cdot}10^6$ parameters [77], and ResNeXt-50 (2017) $25{\cdot}10^6$ parameters [84]. From the NLP domain: BERT (2019) $340{\cdot}10^6$ parameters [13], and GPT-3 (2020) $175{\cdot}10^9$ parameters [5].

[2] The same as two average North-American lives.

[3] To view more approaches, see [20] for a review on pruning techniques, and the related work section [51] for other approaches.

During the past five years, there has been a growing interest to make the DNNs smaller by pruning [20]. In 2019 alone, several techniques [14, 16, 19, 57, 61, 71, 86] have been proposed to remove 60-90% of the trainable parameters without a significant reduction in the model performance. In 2020, lossless compression of neural networks [73] and other promising contributions [54, 68, 82] have already been made to the pruning literature.

Sparse neural networks with random sparsity[1] outperform dense models that have similar number of parameters [14, 52, 91]. Networks with optimized sparsity[2] reach, and sometimes exceed, the performance of the fully parameterized, unpruned models [16, 18, 51, 71, 82]. The good performance of sparse architectures has been commonly recognized [12, 14, 16, 19, 20, 29, 57, 61, 71, 86, 91], but the explanation why the sparse models perform so well is still lacking.

Two explanations (in various forms) are presented in the literature: models with optimized sparsity can exploit redundancy in the data [14, 51], and pruning itself can be equivalent to training the parameters of the network [55, 90]. The latter proves the Lottery Ticket Hypothesis[3] (LTH) [18] and its refined form, LTH with rewinding [19]. However, both of the explanations apply only to models with optimized sparsity, and therefore do not explain why models with random sparsity outperform dense models with similar number of parameters. To understand the success of sparsity more holistically one can leverage the techniques used in the field of explanatory artificial intelligence (XAI) [21].

XAI aims to make the machine learning models more transparent in order to identify problems and have algorithmic fairness. In the context of neural networks, this means understanding data processing and representations inside ANNs. Many of the existing techniques [2, 3, 35] rely on visualizing convolutional filters learned by the network [88]. This approach only works in the visual domain with architectures that include convolutional filters.

While convolutional neural networks (CNNs) [24] are the work horse of the visual domain (image and video) [39], different architectures are used in other domains. For example, in NLP recurrent neural networks (RNNs) [24] are the more widely used alternative [9, 48]. Because of this, often the techniques that make CNNs more interpretable [35, 88] cannot be applied in other than visual domain.

[1] Random sparsity refers to models which have been pruned randomly, without any criteria to choose the pruned weights.

[2] In contrast to random sparsity, in optimized sparsity pruned weights are chosen with some (saliency) criteria.

[3] The LTH claims that a "dense, randomly-initialized, feed-forward networks contain subnetworks ("winning tickets") that - when trained in isolation - reach test accuracy comparable to the original network in a similar number of iterations" [18].

In contrast to convolutional layers, feed forward neural networks (FFNN) are used as a construction block in almost any kind of neural network. CNNs [46] and generative adversarial networks (GANs) [25] used in the image domain, and RNNs used in the natural language domain [9, 58] all have a feed forward network as a part of them. Also, many of the less widely used architectures, like binarized neural networks (BNNs) [10] or capsule networks [32, 70], have a feed forward network as a part of their architecture. Since FFNNs are so widely used, techniques that make them more interpretable have applications in practically any domain.

There is a line of work for the interpretability of FFNNs using the network's activation patterns (APs) [8, 15, 26, 41, 44, 45, 56, 81]. An activation pattern is a collection of neuron activation statuses, e.g. on/off with the ReLU activation function, for each neuron in the FFNN [31, 65]. Simply put, the aforementioned methods observe the activation statuses of neurons to access the logic of feed forward networks [26].

These existing methods observe activations *inside* the networks, which prevents using these approaches to compare networks with different architectures. If two FFNNs have different number of layers, and/or their layers are of different width, their activation patterns are not comparable with each other. This is a serious limitation which prevents the comparison of sparse and dense architectures that have similar number of parameters, because the models have different number of neurons as well. However, this problem can avoided by viewing activation patterns of the network as activation regions (ARs) in the input space.

An activation region is a convex subspace of the input space [31], which corresponds to one unique activation pattern of the network. The network creates a distributed partitioning [4, 48] in its input space by defining a collection of bent hyperplanes [30, 74], where each neuron defines one hyperplane. Subfigures 1.1.a and 1.1.c show input space partitionings over a two dimensional (2D) subspace for two networks with the same number of hidden network parameters[1] (HNPs). Intuitively, activation regions are the connected components [31, 60] between the bent hyperplanes, i.e., the colored regions in the subfigures 1.1.a and 1.1.c.

In the field of neural network expressivity, there is on-going research to estimate the number of activation regions a neural network can create in its input space [30, 60, 63, 65]. Using the tools developed in the network expressivity research, one can view neuron activations inside the network as unions of activation regions in the input space. With this approach the properties of sparse and dense architectures can be compared while leveraging the activations of the networks.

[1]The term *hidden network* refers to all the layers before the output layer. Hidden network parameters are the trainable parameters of the hidden network, i.e., weights and biases of the hidden layers.

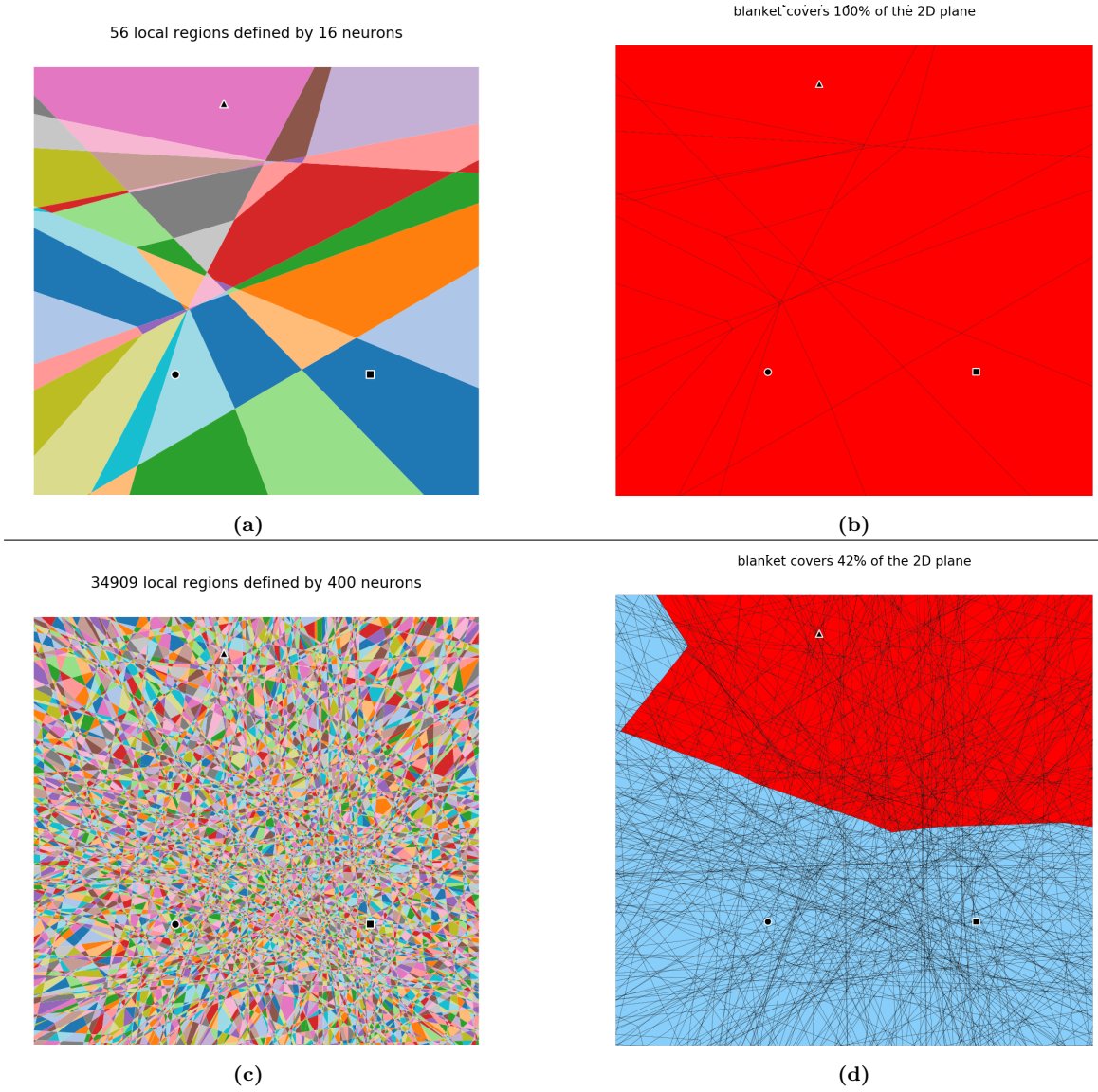


Figure 1.1: Two models that have 9472 parameters in their hidden networks have been trained to recognize fashion items with the MNIST Fashion dataset. On the first row there is a dense network with 16 hidden neurons, and on the second row a sparse network with 400 hidden neurons. In the first column both models split a 2D plane in the input space to activation regions. The plane is spun by three images, one from each class 0, 1 and 9. The images are marked with a circle, square, and triangle, respectively. In the second column the networks define *minimal blankets* w.r.t. class 9. A minimal blanket is a union of activation regions (highlighted red) which activates the subnetwork that is specialized to the phenomenon. The MBH algorithm uses the hypervolume of the minimal blanket to compute the specialization of the network.
(a) A fully parameterized network, density=1.0, with 16 hidden neurons (12,4) splits a 2D plane to 56 regions.
(b) The dense network does not specialize to recognize the class 9, but defines a *trivial blanket*, which covers the whole input space.
(c) A sparse network, density≈0.034, with 400 hidden neurons (LeNet 300,100) splits the same 2D plane to 34 909 regions.
(d) The minimal blanket defined by the sparse network covers 42% of this 2D subspace.

Leveraging the connection between APs and ARs is not enough to make FFNNs interpretable by itself. Using activation regions to analyze FFNNs is problematic, since the computational requirements to precisely construct an input space partitioning are intense: the exact input space partitioning for a network with 22 hidden neurons can take over 30h to compute [74].

In this work I explain the successes of random and optimized sparsity by analyzing the distributed partitionings that networks create in their input spaces. My approach is compatible with the existing explanations, and in addition it offers an intuitive explanation why networks with random sparsity perform better than dense networks with the same number of hidden network parameters.

A sparse model defines more hyperplanes in the input space than a dense model with the same number of HNPs, because each neuron defines exactly one (bent) hyperplane in the input space. Defining more hyperplanes enables the model to split the input space to more (activation) regions [87]. In Figure 1.1 the sparse network splits the 2D plane to over 500 times more activation regions than the dense network, regardless of both having the same number of HNPs.

Defining more regions has been associated with the network's ability to approximate a more complex function [30, 60, 63, 65, 66, 74], and therefore sparse networks can approximate more complex functions than dense networks with the same number of HNPs. As seen in figures 3.4 and 3.5, models that define more hyperplanes (sparse) perform better than models that define less hyperplanes (dense). In other words, quantity over quality with hyperplanes.

While defining more hyperplanes enables the model to approximate more complex functions, it is not the only potential benefit from distributing parameters over larger number of neurons. As discussed earlier, the ability to learn concepts on different levels of abstraction is the core of the deep learning paradigm. My thesis is that having larger number of neurons enables the network to specialize to the features in the data more distinctly.

As my main contribution, I define **network specialization**, accompanied by a specialization measure to evaluate it, and an algorithm to compute it. Network specialization is a novel concept, which considers how distinctly a feed forward neural network has learned to recognize high level features in the data. Intuitively, if a feed forward network F is specialized to recognize images (image domain) or singing (audio domain) of a certain species of birds, then it has a subnetwork S that is activated *only* by samples (image or audio) that represent the bird species it is specialized to.

I present **Minimal Blanket Hypervolume (MBH) algorithm** to compute

the specialization measure for a FFNN. It uses the network's decision patterns[1] to identify the subspace, which the network associates with a given high level feature.

The algorithm uses a greedy heuristic to find the smallest subnetwork, which is activated by inputs with the high level feature. The computationally prohibitive construction of the full input space partitioning is avoided by computing the partitioning on a lower dimensional subspace. This way the partitioning is computed precisely, which guarantees that no activation region in that subspace goes unnoticed. By averaging over randomly chosen subspaces, and using the same subspaces to measure the specialization of different networks, the MBH algorithm approximates the full input space partitioning in reasonable time, regardless of the exponential time complexity of the problem.

In subfigures 1.1.b and 1.1.d, a 2D plane in the input space is split into regions by two networks with the same number of HNPs. The color of region indicates if it belongs to the subspace which the network associates with the high level feature: red regions are associated, blue regions are not.

Because MBH evaluates the distributed partitioning in the input space, which is shared between the networks that solve the same problem, it can be used to compare networks with any number of hidden layers of arbitrary width. It is also training agnostic, meaning that it does not need to be present during the training of the network, but can be applied to any trained network.

With MBH one can numerically evaluate the specialization of any FFNN, in any domain. Only a dataset with labeled high level features is needed: for example, in datasets designed for supervised learning, some high level features have been labeled as classes. The dataset can be different from the training or validation data used to train the model, which enables evaluating how specialized the model is w.r.t. any concept that can be embodied in the data.

While network specialization can in part explain why sparse networks perform so well, it can also make the networks more interpretable. MBH provides a tool to evaluate how tightly a FFNN outlines any user defined abstract concept in the input space; an insight that directly communicates how well the network expresses the core concept of deep learning.

I test my hypotheses by comparing sparse and dense models in two scenarios: 1) sparse and dense models with the same number of HNPs, and 2) sparse and dense models that have the same architecture[2]. In the first scenario the sparse networks are

[1] A decision pattern is a part of the full activation pattern of the network; it specifies the activation statuses of some subset of neurons [26].

[2] When comparing different FFNNs, by "architecture" I refer to the number of neurons and how they are distributed to layers. Sparse and dense models that share the same architecture have different

pruned at random, and therefore the effort of initializing a sparse network is comparable to the dense counterparts. In the second scenario sparse networks have optimized sparsity, which compensates for the loss of trainable parameters compared to the dense (fully parameterized) models. For example, comparing the "lucky" subnetworks (winning tickets in the LTH [18]) to the fully parameterized, dense networks fall under the second scenario.

The scope of this work is limited to the image domain, and piecewise linear feed forward neural networks with ReLU activation function. Because of the limited space of this work, I evaluate models on MNIST Digit and Fashion datasets, leaving the more comprehensive evaluation for future work. It is good to note that even though this work mainly considers the image domain, network specialization can be applied to any domain where architectures include feed forward networks.

To summarize, the contributions presented in this work are

- **Network specialization**, a new concept that considers how distinctly the hidden network of a FFNN has learned to recognize some abstract, user defined concept (sections 4.1 and 4.2, Definition 4.5).
- **Specialization measure** to evaluate, how specialized a FFNN is (Section 4.3, Definition 4.7).
- **MBH algorithm** to compute the specialization measure in a reasonable time, regardless of the exponential time complexity of the problem (Chapter 5, Algorithm 1).
- **Intuitive explanation for the performance of sparse networks** by viewing neural networks as collections of bent hyperplanes (Section 3.3). This includes the ability to approximate more complex functions (Subsection 3.3.1), and to specialize more, both evaluated with experiments in the image domain (Chapter 6).

To build the background for network specialization, I will start by reviewing existing literature and defining relevant concepts. In Chapter 2, I will cover sparsity, and in Chapter 3 activations in artificial neural networks. In Chapter 4, I will introduce network specialization, and in the following Chapter 5 I propose MBH algorithm to measure it. The algorithm will then be used in the experiments of Chapter 6 to evaluate my hypothesis that sparse networks specialize more to high level features than dense networks with the same number of hidden network parameters.

number of non-pruned weights, but the same number of neurons, distributed to layers in the same way.

2. Sparsity in Artificial Neural Networks

Deep neural networks are powerful models with often tens of millions of parameters [5, 20, 46], which causes the models to have a big memory footprint and single inferences that require a billion memory accesses and arithmetic operations [91]. These resource requirements are often prohibitive, especially when the models are used in edge devices such as mobile platforms, wearable devices, or smart health devices [54, 91]. Sparsity has been offered as an answer to these problems over the past decade, while the concept of pruning network weights for better generalization and quicker training dates back to the end of 1980's [50].

A sparse neural network has some of its parameters removed, or otherwise made inactive [20]. A sparse network with some fixed architecture A has therefore less (active) weights than a dense network with the same architecture. Sparsity of a neural network means the following in this work:

Definition 2.1. Sparsity
Let F be a feed forward neural network with l layers. Each layer has a weight matrix W_l to store the weight parameters of that layer. The network F is *sparse*, if a significant[2] number of the entries in the weight matrices are zeroes.

The opposite of a sparse network is a dense network. In a dense network an evident majority of the entries in the weight matrices are non-zero. If a FFNN F has only non-zero parameters, then its density $d = 1$. If 90% of the trainable parameters[3] are removed or set to zero, then the density of F is $d = 0.1$.

A network can be incentivized to be sparse by adding L_1 or L_2 norm of the weights to the loss function [29], which is called L_1 or L_2 regularization, respectively. Another way to introduce sparsity is dropout [34], where some randomly chosen entries

[2]There is no absolute measure when a matrix is sparse. In the context of neural networks, removing 100 entries from a 300×100 matrix (density ≈ 0.997) would not make the matrix sparse. Removing 100 entries from a 20×10 matrix (density 0.5) would be considered a sparse matrix.

[3]In the case of FFNNs trainable parameters are the weights and biases of the network.

are treated as zeroes during each forward pass during the training of the network. In this work, however, I will concentrate on sparsity achieved by *pruning*. Pruning a network means removing or otherwise disabling entries from the weight matrices of the network.

While there has been several successes in reducing the memory footprint and computational requirements of dense models, while maintaining comparable accuracy [29, 54, 73, 91], the field is under constant development. It is far from clear how the networks should be pruned, and what is important for the performance of a sparse network [19, 20, 52].

In this chapter, I will introduce important concepts related to network sparsity, and take a more detailed look at the Lottery Ticket Hypothesis: a conjecture on the existence of trainable sub-networks in dense, overparameterized networks [18]. I begin by defining some core concepts related to ANNs in Section 2.1. In Section 2.2, I define and explain network pruning as means to obtain sparse neural networks. I conclude by bringing together the benefits of optimized sparsity in Section 2.3.

2.1 Preliminaries

In this section I define the core concepts related to artificial neural networks (ANNs) that are essential preliminaries for the rest of this work.

ANNs [24] are connected collections of computational units called neurons that are usually organized in layers. For the purpose of this work, a neuron can be defined as follows:

Definition 2.2. Neuron [24, 31]
Let n_i be a neuron that computes a scalar output y_i,

$$y_i = g(a_i) \tag{2.1}$$

$$a_i = f_a(x_i; w_i, b_i) = \sum_{j=0}^{d_l} x_{ij} w_{ij} + b_i \,, \tag{2.2}$$

where d_l is the input dimension of the neuron, $a_i \in \mathbb{R}$ is a pre-activation that is a dot product between the input vector $x_i \in \mathbb{R}^{d_l}$ and neuron's weight vector $w_i \in \mathbb{R}^{d_l}$ plus the bias term $b_i \in \mathbb{R}$, and $g(\cdot)$ is an activation function.

For the network to learn a non-linear input distribution, the activation function $g(\cdot)$ needs to be non-linear [24]. In this work, I will concentrate on networks that have rectified linear unit (ReLU) as their activation function, formally $g(x) = max(0, x)$ [27].

In the case of ReLU, the non-linearity results from the different treatment of negative and positive values. When $x \geqslant 0$ the pre-activation stays unaffected; $g(a) = a \iff a \geqslant 0$. If the pre-activation is negative, however, the value is rectified to 0; $g(a) = 0 \iff a < 0$.

In this work, I mainly consider traditional feed forward neural networks (FFNN), which feed the input through the network one layer at the time using the output of one layer as the input of the next layer. The feed forward layer can be defined as

Definition 2.3. Feed Forward Layer following [60]
Let $\mathbf{n}$ be a set of k neurons, which all have the input dimension of d_0. Together the neurons form a feed forward layer $f_l(\cdot)$, which maps its input $\mathbf{x}$ to an output $\mathbf{y}$ by applying all of the layer's neurons to the input and then concatenating the neurons' outputs to a vector

$$\mathbf{y} = f_l(\mathbf{x}) = [\, g(f_a(\mathbf{x}; w_1, b_1),\, g(f_a(\mathbf{x}; w_2, b_2),\, \ldots,\, g(f_a(\mathbf{x}; w_k, b_k)\,], \tag{2.3}$$

where $\mathbf{x} \in \mathbb{R}^{d_0}$, $\mathbf{y} \in \mathbb{R}^k$, and g and f_a are as defined in equations 2.1 and 2.2, respectively.

As seen in Equation 2.3, all the neurons in a feed forward layer share the same input $\mathbf{x}$, but each neuron applies linear transformation to the input with their own weights and biases. The linear transformations are passed through the activation function $g(\cdot)$, and those outputs concatenated as the output of the layer.

When several feed forward layers are stacked on top of each other, so that one layer's output is another's input, the construction is called a feed forward neural network. Following [60]:

Definition 2.4. Feed Forward Neural Network following [60]
is a composition of l feed forward layers. It defines a function $F : \mathbb{R}^{d_0} \to \mathbb{R}^m$, where d_0 is the dimension of the input space and m is the dimension of the output space. The function F is of form:

$$F(x; \Theta) = f_{out} \circ f_l \circ f_{l-1} \circ \cdots \circ f_1(x), \tag{2.4}$$

where each layer maps its input x_l to an output y_l. Input x_l is the output of the previous layer $x_l = y_{l-1}$ for all layers other than the input layer, for which it is the input of the network.

The last layer of the network is known as the output layer. A FFNN can be divided to two parts: the output layer, and all other layers before it. Since the user only observes the output layer of the network to know the output of the network, the layers preceding the output layer are also known as the *hidden layers* of the network.

It is important to note that the layer which outputs the class or regression decision of the network never accesses the input data directly. The signal and the noise in the data are processed by the hidden layers, with the intention to extract the signal that is important for the decision of the output layer. Therefore the hidden layers, also known as the *hidden network*[1], can be seen as a feature extractor that prepares the input data for the output layer.

If the activation function g is linear, then the network will collapse to a single linear function, and it does not learn non-linear phenomena [24]. When using a non-linear activation function, however, even the simple feed forward architecture can approximate complex, non-linear functions to an arbitrary precision: feed forward networks can approximate any Lebesgue integrable function, and are therefore universal function approximators [36].

An example of a function that one might be interested to approximate could be classifying if a patient has cancer or not, based on their medical records. In this example the input space is the medical record, where each entry is a value of one input dimension, e.g. 70kg could be the value of a dimension "weight of the patient". The output space could be one dimensional: a floating point number representing the probability that the patient has cancer. Because of the universal approximation theorem, it is certain that *if* the function that reliably maps a medical record to cancer diagnose exists, there exists FFNN that approximates it precisely.

Neural networks do not necessarily approximate any interesting function at the initialization of the network. The parameters of the network (weights and biases of Equation 2.2 in the case of FFNNs) are drawn from some distribution when the network is initialized. It is highly unlikely to draw parameters that would right away approximate some function the user is interested in approximating. Therefore the initialized parameters have to be changed for the network to approximate the target function.

Changing the values of the parameters of the network is called *training* the network. Data that represents the behaviour of the target function, gradient based methods, and the back propagation algorithm are used to train the network to approximate the target function better [24]. Even though the universal approximation theorem says that there *exists* a set of parameters with which a sufficiently large network can approximate any target function, it does not say that the optimal parameters can be found.

How to train and optimize neural networks to approximate the target function is the core question in neural network research [24]. One aspect of it is the architecture of the networks: some architectures are better suited to learn to approximate a given target function than others. For example, some problems are easier to approximate

[1]The hidden layers of a FFNN are a FFNN by themselves.

with deep architectures, where the network is constructed from many narrow layers instead of few wide layers. This approach is also known as *deep learning*, which has provided good results in several problem domains [48].

FFNNs have been known to be universal function approximators for more than 30 years. Recently it has been shown that depth-bounded [47] and width-bounded [53] rectified networks[1] are universal approximators as well. Since ReLUs were applied to stabilize the training of deep neural networks in 2011 [22], ReLUs have become widely used in deep learning [31, 44, 47, 74, 80].

2.2 Pruning

Pruning means removing a network's weights, usually by setting them to zero [20]. Traditionally in machine learning, a model which has more parameters is expected to be more powerful, i.e. is able learn more complex target functions. However, there is a growing body of work [11, 12, 14, 16, 18, 28, 29, 38, 51, 52, 54, 57, 61, 68, 71, 73, 82, 83, 86, 90] showing that this expectation does not hold true on pruned neural networks: removing trainable parameters (weights) of the networks does not significantly hurt the models' performance, and in some cases makes the networks to perform better than the original, unpruned models.

There are several strategies for which weights ought to be pruned and when. Simply put, one can prune weights either before, during, or after training. Corresponding pruning strategies are *pre-defined sparsity* [14], *iterative pruning* [18], and *traditional pruning* [29], respectively.

Another aspect to tell pruning strategies apart is to consider wether weights are pruned in a *structured* [54, 83] or *unstructured* [29, 91] manner. The former prunes collections of weights, such as whole filters or convolutions in a CNN; the latter means pruning weights individually, without paying attention to any bigger structures of weights.

A third choice to make regarding pruning is which weights are pruned, i.e. define the *pruning criteria*. Magnitude pruning is the most common choice [20], although other strategies have been proposed as well [85, 90]. The weights can be pruned without sophisticated pruning criteria, in which case the weights are pruned at random [14]. In the context of this work, *random sparsity* refers to networks that are pruned at random, and *optimized sparsity* to networks that have been pruned with a more sophisticated pruning criteria.

The fourth aspect of pruning to consider is the finality of a pruning decision: can

[1] Rectified network uses solely ReLUs as its activation function. All the networks examined in this work are rectified networks, if not specified otherwise.

a once pruned connection be revived later in the pruning process? When the network topology is also optimized, i.e. pruned weights can be "revived" throughout the training process, pruning is *dynamic pruning* or *dynamic sparsity* [16, 17, 29, 71].

The four different aspects of pruning weights of a neural network are presented in Table 2.1.

Aspect of Pruning	Approaches
Timing	Before / During / After training
Granularity	Structured / Unstructured
Criteria	Optimized / Random
Finality	Pruned weights can / cannot be revived later in the process

Table 2.1: A simple framework for classifying neural network pruning strategies regarding the timing, granularity, pruning criteria, and finality of the pruning.

In the later chapters of this work, I will concentrate on examining networks that are randomly pruned prior to the training. Benefits that follow from sparsity obtained in this manner can be credited to the sparsity itself, and not other aspects of the pruning method.

Before proceeding to study the inherent benefits of sparsity in neural networks in the following chapters, I will examine a phenomenon related to optimized sparsity.

2.2.1 The Lottery Ticket Hypothesis

Recently Frankle and Carbin conjectured that large, overparameterized FFNNs contain subnetworks, which have won the initialization lottery [18]. They argued that the existence of "lucky initializations" (winning tickets, WTs) would explain why the training of a large network converges successfully with a higher probability than the training of a smaller network. The large, overparameterized network contains many subnetworks, "lottery tickets", making it more probable that at least one of them is a winning ticket which trains well.

When trained in isolation, these winning tickets perform better than randomly initialized networks that have the same sparsity, i.e. normal lottery tickets. Winning tickets can be trained to the same test accuracy as the original dense network, regardless of WTs having potentially less than 10% of the parameters of the dense network.

The hypothesis, as presented in the original work, is stated as follows:

Conjecture 2.1. The Lottery Ticket Hypothesis [18]
A randomly-initialized, dense neural network contains a subnetwork that is initialized

such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.

The default method to to obtain winning tickets (WTs) goes as follows:

1. Initialize a dense, overparameterized network.
2. Train the dense network until it converges.
3. Prune the network to some sparsity s. This is the structure of the winning ticket.
4. Take the pruning mask from 3 and return weights to their initial values from 1. This is the winning ticket.

Furthermore, the winning tickets are usually found through iterative magnitude pruning (IMP) because iterative pruning produces better sparse networks [18, 29]. It is good to note, however, that the conjecture does not require this, since it doesn't state anything about the pruning technique used to find these "well" initialized subnetworks.

The lottery ticket hypothesis (LTH) has gained a lot of attention during 2019 and 2020 [11, 16, 19, 52, 55, 57, 61, 68, 71, 79, 80, 82, 86, 90]. For the purpose of this work it is interesting for two reasons. Firstly, the existence of winning tickets (i.e. well performing subnetworks) is interesting in itself, since it means that potentially all dense networks could be pruned for a better efficiency and performance. Secondly, the results in the literature are in some parts contradictory, so clearly there remains work to be done on the topic.

There is contradicting evidence on the role of winning tickets. Most of the critique is directed towards the role of winning tickets: for some datasets and optimization schemes winning tickets "do not exist", or are not distinguishable from random initializations after substantial amount of training [20, 52]. This implies that winning tickets do not play a central role in the trainability of large neural networks, unlike the authors of the conjecture originally hypothesized.

2.3 Pruning Can Be Training

The contemporary approach to teach a network to approximate a function, i.e. to solve a problem, is to change the learnable parameters of the network with gradient based methods. The goal is to find a good set of parameters which solve the problem. However, fine-tuning the network's parameters is not the only way to teach the network to approximate some function.

Pruning the network's weights can be training [90]. In fact, a sufficiently large piece-wise linear network can be pruned to approximate any (Lebesgue integrable)

function without changing the unpruned weights at all [55]. This contributes to the success of sparse networks with optimized sparsity: a successful pruning might teach the network to solve the problem without training the weights. This can be seen in Figure 2.1, as well as in [90].

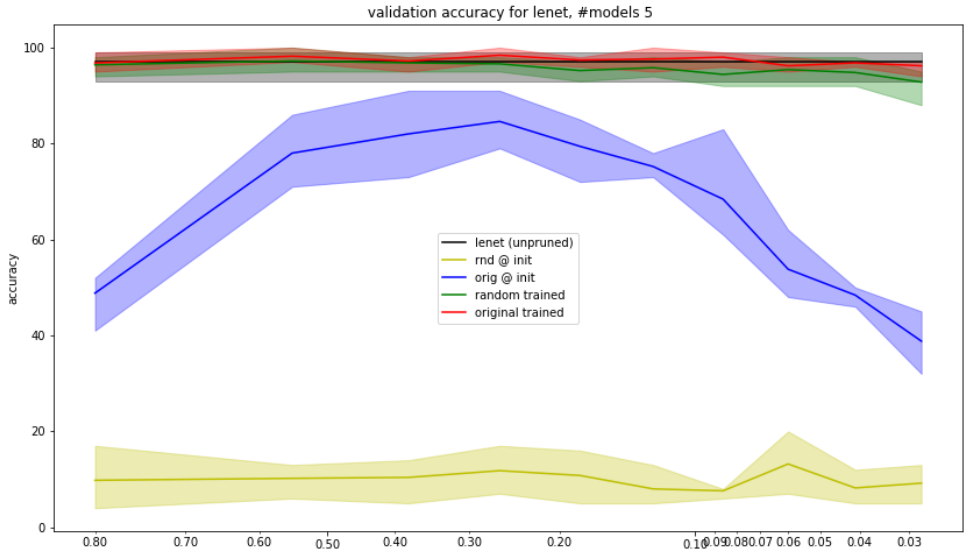


Figure 2.1: Neural networks can be taught to solve a problem without changing the weights. Validation accuracy of Lenets with (300,100) hidden neurons on Digit MNIST, averaged over 5 models. Intervals are min and max. Blue and yellow have untrained parameters. Blue (winning tickets) are obtained from the fully parameterized models (black) as described below Conjecture 2.1. In short, the WTs have the pruning mask obtained from the trained fully parameterized models, but the weights are from the initialized state, before the training. The pruning is done with the one-shot magnitude pruning. Yellow has random pruning and re-initialized parameters. Red marks the trained winning tickets that is, red is blue after training. Green is yellow after training. The x-axis is density.

As explained in the Chapter 3, the weights of a neuron define the orientation of the hyperplane it spans in the input space. Both removing some of the weights and changing the values of the weights through training alter the orientation of that hyperplane. Therefore it is not surprising that, by removing some of the weights, the hyperplane can be adjusted into a favourable position for the purpose of solving the problem at hand.

In Figure 2.1 a vastly overparameterized network ($300 + 100 = 400$ hidden neurons) can achieve consistently over 80% validation accuracy on a simple image classification problem without training the weights, just removing 70-75% of them. Figure 2.1 also demonstrates the Occam's hill [67] produced by pruning: by removing too few or too many weights (left and right edges of the plot), the network does not achieve a high validation accuracy. By removing enough, but not too many, of the weights the hyperplanes are adjusted enough to (partially) solve the problem without changing the

values of the weights at all.

Both the winning tickets (blue) and randomly pruned, re-initialized networks (yellow) in Figure 2.1 reach rather similar validation accuracy (red and green, respectively). This supports the view presented in [20, 52] that randomly pruned, freshly initialized networks can be trained to the same validation accuracy as the winning tickets. Strong conclusions cannot be drawn, however, since winning tickets found with one-shot magnitude pruning are reported to be significantly worse, than ones found with iterative magnitude pruning [18, 90].

Training the network by removing weights is not the only benefit of a well chosen pruning mask. Removing connections can serve as a way to reduce the noise without disturbing the signal, enabling the model to *exploit the redundancy* in the data [14].

2.3.1 Exploiting Redundancy in the Data

Sometimes the data have dimensions which do not hold any information about the phenomena that the user is modelling, i.e. the dimensions are redundant. In the case of hand written MNIST digits this is clearly visible. The pixels at the borders of the images do not tell anything about the digit in the image because the digits are centralized and do not reach the edges of the image. Since those dimensions (pixels near edges) do not have any information about the phenomena (hand written digits) that is being modelled, they only introduce noise to the input of the network. Removing the connections to these dimensions improves the signal to noise -ratio of the input that the network receives.

Stochastic gradient descent (SGD) does this naturally, without any further incentivising[1]. In Figure 2.2 the right-hand column shows learned pruning masks obtained with one-shot magnitude pruning (OSMP). The more weights are pruned, the more dimensions near the border of the images are left completely without connections. This means that after training with the SGD, weights to those uninformative dimensions are so small that they get pruned by the OSMP, and the more informative connections are left for the model to be processed. In other words, pruning a network with a suitable pruning criteria can serve as a feature selector.

If the data are highly redundant, i.e. there are a lot of uninformative or repetitive dimensions, then pruning can improve the quality of the data received by the network by removing connections to these uninformative dimensions. Sparse architectures have been successful especially in tasks with highly redundant data, discussed for example in [14]. This suggests that one factor contributing to the success of winning tickets is the

[1]In this experiment scenario with Digit MNIST.
Exhaustive proof is left for future work.

ability to prune noise from the input signal, rather than having a "lucky initialization", like the authors of LTH originally argued [18].

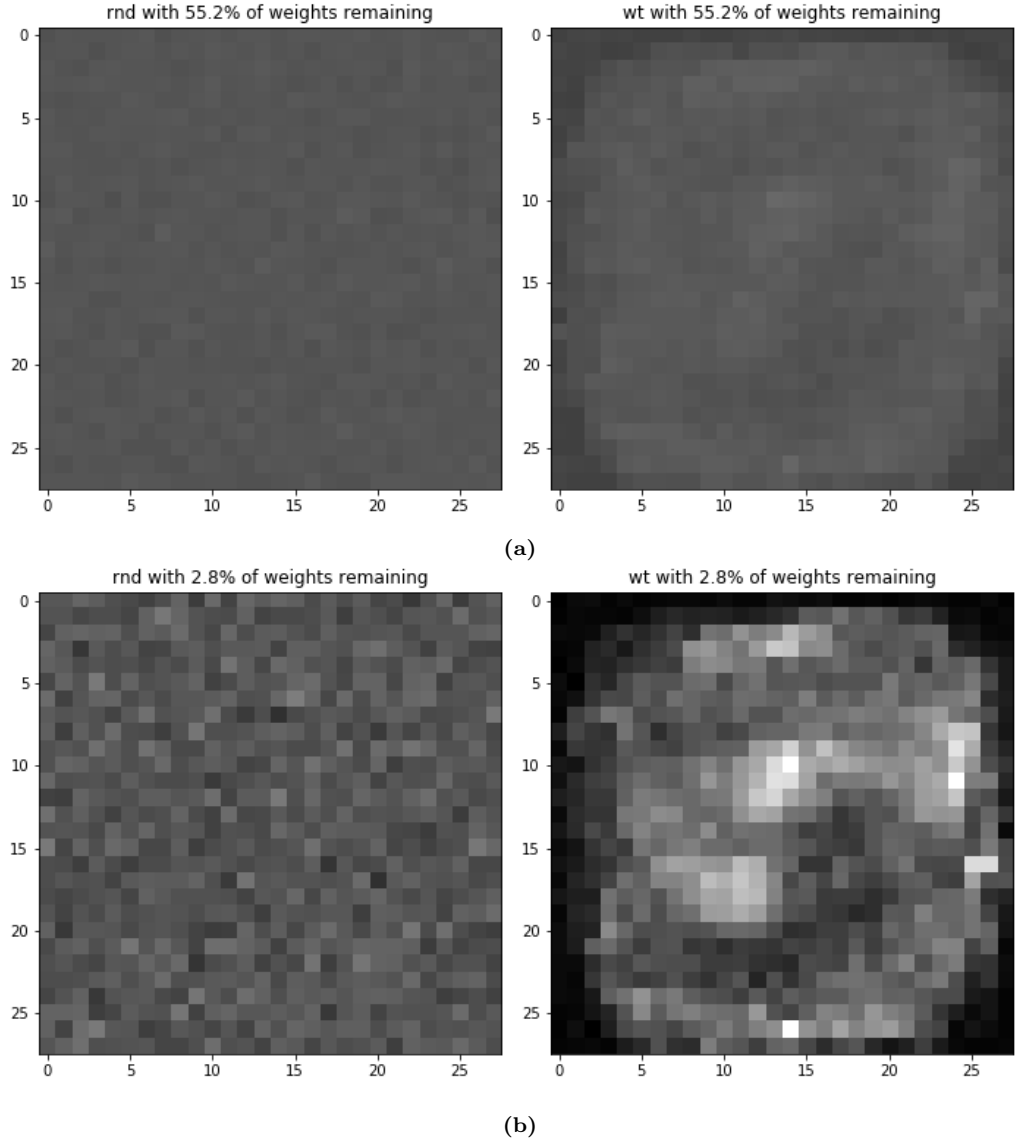


Figure 2.2: Heatmaps of active connections on the input layer of Lenet (300,100 -hidden neurons) after pruning. A white pixel indicates, that every neuron in the input layer has unpruned connection to that dimension; a black pixel means that all the connections have been pruned. Each heatmap is the sum of five networks. In (a) the density of the pruned networks is 0.552, in (b) the density is 0.028. Left: random sparsity. Right: optimized sparsity (OSMP).

3. Activations in Piecewise Linear Neural Networks

When a neural network processes data, some of its neurons are activated by the input. Different inputs activate different neurons, and some neurons are more often active than others. The study of activations in neural networks examines the activation statuses of the neurons to understand the networks and their behaviour better.

Over the past 6 years there has been an increasing interest towards activations in neural networks to answer questions on expressivity [60, 66] and interpretability [26, 44] of FFNNs. I will extend on the tools developed on this field in Chapter 4 to define and measure the specialization of FFNNs.

In this chapter, I define and explain key concepts related to activations in artificial neural networks (Section 3.1), and review how activations are used in the existing literature (Section 3.2). Finally, in Section 3.3, I argue why sparse networks can be expected to specialize more than dense networks with the same number of HNPs.

3.1 Background and Definitions

In this section, I cover essentials on activations inside a network (Subsection 3.1.1) and activations in the input space (Subsection 3.1.2). Before proceeding to the activations, I will briefly define piecewise linear (PWL) networks.

A piecewise linear function consists of several linear functions which apply to collection intervals of real numbers [74]. For example, the ReLU -function[2] is a PWL function that consists of two linear functions defined for intervals $x < 0$ and $x \geqslant 0,\ x \in \mathbb{R}$.

If a feed forward neural network uses ReLU as its activation function, the function that the network expresses is a *piecewise linear* function [22]. Intuitively this means that the network defines a different linear function for each *linear region* of the input space, and the location of an input x in the input space will determine which linear

[2]See Section 2.1 for details on ReLU.

function will be applied to x, when processed by the network.

A piecewise linear network is a neural network, whose activation functions are piecewise linear. Therefore a rectified network is a PWL network. I am restricting my analysis to piecewise linear networks because the existing research on network activations concentrates solely on PWL networks.

3.1.1 Activations in a Neural Network

The atomic unit which can have an activation status in a FFNN is a neuron. ReLUs offer an intuitive example for defining when a neuron is *active*. Since the negative pre-activation a_i results a constant 0 with ReLU, I shall say that a neuron is inactive (or "off") when the pre-activation is negative, and active (or "on") otherwise. In more general terms:

Definition 3.1. Neuron activation status, following [60, 65]
A neuron's activation status changes, when the activation function g has irregular behaviour, such as an inflection point or non-linearity. The number of different activation states of a neuron $\alpha = r_g + 1$, where r_g is the number of irregular behaviours of the activation function.

The set of possible activation statuses $\mathbf{a}$ depends on the activation function. For example, for a neuron with ReLU activation, $\mathbf{a}_{ReLU} = \{0, 1\}$ and $\alpha_{ReLU} = 2$. For hard hyperbolic tangent (hard tanh), $\mathbf{a}_{tanh} = \{-1, 0, 1\}$ and $\alpha_{tanh} = 3$ [65].

We can also consider the activation status of the whole network, i.e. the activation statuses of all neurons simultaneously. The activation status of a network can be expressed as an activation pattern:

Definition 3.2. Activation pattern $\mathcal{A}$ [31, 65]
Let F be a neural network with k neurons, g the activation function of the network, and $\mathbf{a}$ the set of possible activation statuses of g. An activation pattern for F is an assignment to each neuron an activation status [31]:

$$\mathcal{A} := \{a_z,\ z \text{ is a neuron in } F\} \in \mathbf{a}^k. \tag{3.1}$$

Let x be an input and Θ the set of parameters of F. Let $AP(\cdot)$ be a function that maps F, Θ, and x to the corresponding activation pattern $\mathcal{A}$ [65], that is,

$$\mathcal{A} = AP(\, F(x; \Theta)\,)\ . \tag{3.2}$$

As seen in the Equation 3.2, the activation pattern of the network depends both on the input x and the parameters of the network Θ. Changing either of them can change the resulting pattern $\mathcal{A}$.

Because the size of an activation pattern depends on the architecture of the network F, and a different set of parameters Θ can result in a different activation pattern, for the sake of readability we will assume the architecture[1] of F and Θ fixed, if not stated otherwise.

When a neuron's activation status is fixed, the neuron has an *activation constraint*. In an activation pattern, all neurons have an activation constraint. In some applications this is too restrictive, as one might be interested in the behaviour of some particular subset of neurons.

The more fine grained patterns, which consider only a subset of neurons, are called *decision patterns*:

Definition 3.3. Decision Pattern σ [26]
Let F be an FFNN, x an input of F, and $\mathcal{A}$ an activation pattern of F when F processes x. A decision pattern σ is a sub-pattern of $\mathcal{A}$. That is, σ specifies the activation statuses for some subset of neurons in F, when F processes x.

When the subset of neurons coincide with a layer of the network, the decision pattern can be called a *layer pattern*.

Definition 3.4. Layer Pattern $\mathcal{L}$, following [26]
Let F be a FFNN, and L the i:th layer of F. A layer pattern $\mathcal{L}_i$ is a decision pattern σ, for which all the neurons constrained in σ belong to L, and all neurons in L are constrained in σ.

In Figure 3.1 the difference between activation, decision, and layer patterns is illustrated with a simple example.

Decision patterns, and therefore layer patterns as well, outline a *subnetwork* of the FFNN. A subnetwork S is a subset of neurons, which is defined by a decision pattern σ, and it is connected to the input space through neurons that feed into σ. Formally,

Definition 3.5. Subnetwork S
Let F be a FFNN, and σ a decision pattern of F. The subnetwork S, which σ defines, consists of the neurons that 1) belong to σ, or 2) feed into the neurons in σ.

If a neuron's output affects the input of another, the former *feeds* into the latter [26]. If a FFNN is fully connected, then every neuron on a layer l feeds into all neurons on the layer $l+1$. This also extends deeper into the network: if a neuron n_1 on layer 1 feeds into the neuron n_2 on layer 2, and n_2 feeds into the neuron n_3 on layer 3, then n_1 feeds into n_3. For the purpose of feeding, a FFNN is considered as an directed acyclic graph (DAG).

[1] Architecture meaning the size and number of layers.

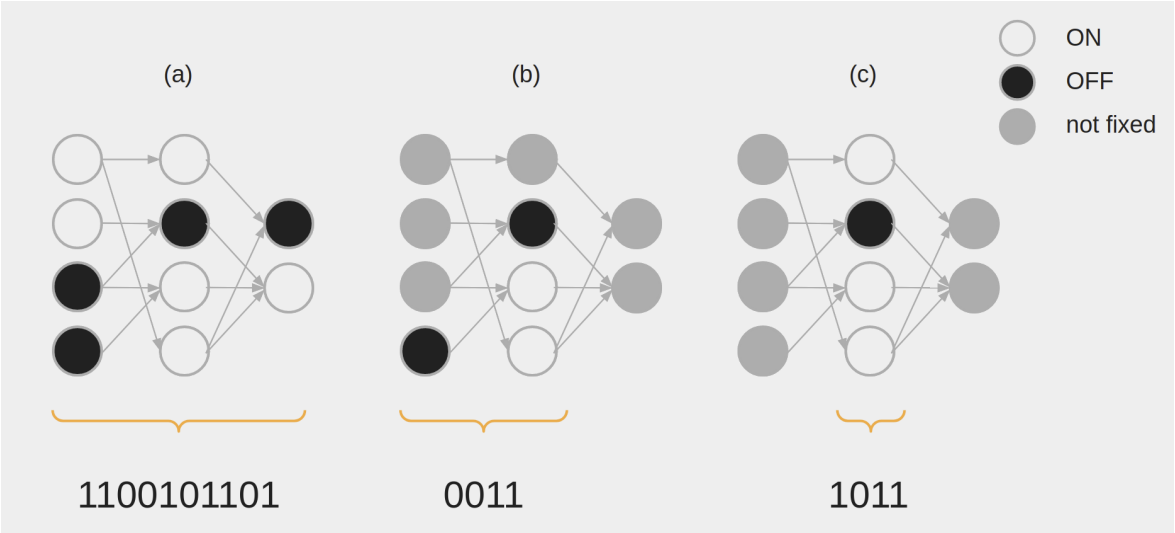


Figure 3.1: Illustration of (a) an activation pattern, (b) a decision pattern, and (c) a layer pattern of a sparse FFNN with (4,4,2) neurons and ReLU as its activation function. Gray nodes do not have an activation constraint, black nodes are required to be inactive and green nodes active, when the networks processes an input. Note that in an implemented version of decision and layer patterns the neuron indices should be saved alongside the activation statuses.

For example, the decision pattern presented in Subfigure 3.1.b defines a subnetwork of seven neurons: all four neurons form the first layer, and the three constrained neurons from the second layer.

A subnetwork is activated by an input x, when the processing of x results in the same decision pattern which defines the subnetwork. If a subnetwork S of a FFNN F is defined by a decision pattern σ, then all the neurons constrained in σ should have the same activation status when F processes x, for S to be active.

The activations inside the neural network can be used to improve the interpretability and performance of the networks, as seen later in Subsection 3.2.2. However, the usage of activations is not limited to neurons inside the networks: every activation inside the network can be interpreted as a region in network's input space.

3.1.2 Activations in the Input Space

A neural network creates a distributed[1] partitioning [4, 48] in its input space. Inputs located in different parts of the partitioning result in a different activation pattern

[1]In a *distributed* partitioning, a point in the input space is classified by each hyperplane spun by the model, e.g. each neuron of a neural network. A model that creates a *local* partitioning, e.g. a k-means clustering, connects each input to a local region in the input space, rather than making a binary classification with each computational unit [4]. Because of this the *knowledge representation* learned by a neural network is said to be distributed.

when they are processed by the network[1] [26, 31, 30].

As discussed in the beginning of this section, a PWL neural network defines different linear functions for the different parts, or *regions*, of the partitioning. When moving from one linear region to another in the input space, the linear function changes as one or more of the activation functions g change from one linear state to another. The boundary between two or more linear regions in the input space is a *hyperplane*.

Definition 3.6. Hyperplane H [60, 63]
Hyperplane H_i is a subspace in the input space:

$$H_i := \{x \in \mathbb{R}^{d_0} \mid f_a(x; w_i, b_i) = 0\}. \tag{3.3}$$

It is spun by a neuron n_i with the weight vector w_i, bias b_i, and ReLU as the activation function. The pre-activation function f_a is defined in Equation 2.2.

For example, consider a two dimensional input space $d_0 = 2$ and a neural network with ReLU activations. Each neuron on the first layer of the network "draws a line" in the input space, and the line is located where the neurons pre-activation a_i changes its sign. That line is the hyperplane a neuron defines in the input space.

A collection of all hyperplanes of the first layer of the network (one for each neuron) is called a *hyperplane arrangement* [60, 75]. A hyperplane arrangement consists of linear hyperplanes, i.e. the hyperplanes do not *bend*.

However, hyperplanes defined by neurons that are deeper in the network can bend. The inputs received by neurons after the input layer have passed through the non-linear activation function g, and therefore the inputs can be defined by different linear functions depending on the interval of the original input.

The set of hyperplanes defined by a neural network with has several layers is referred to as a *collection of bent hyperplanes*:

Definition 3.7. Collection of Bent Hyperplanes $\mathbf{H_F}$ [31, 60]
A collection of bent hyperplanes is a set of hyperplanes

$$\mathbf{H_F} := \{H_1, H_2, \ldots, H_n\}$$

in the input space, spun by a network F that has n neurons.

In Figure 3.2 the collection of bent hyperplanes defined by a small DNN is visualized to illustrate the bending of hyperplanes.

Each hyperplane splits the input space to two regions: one where the spanning neuron is active, and another where it is not. Since the activation statuses of neurons

[1]This equals to changing the input x in Equation 3.2, while keeping the networks parameters Θ fixed.

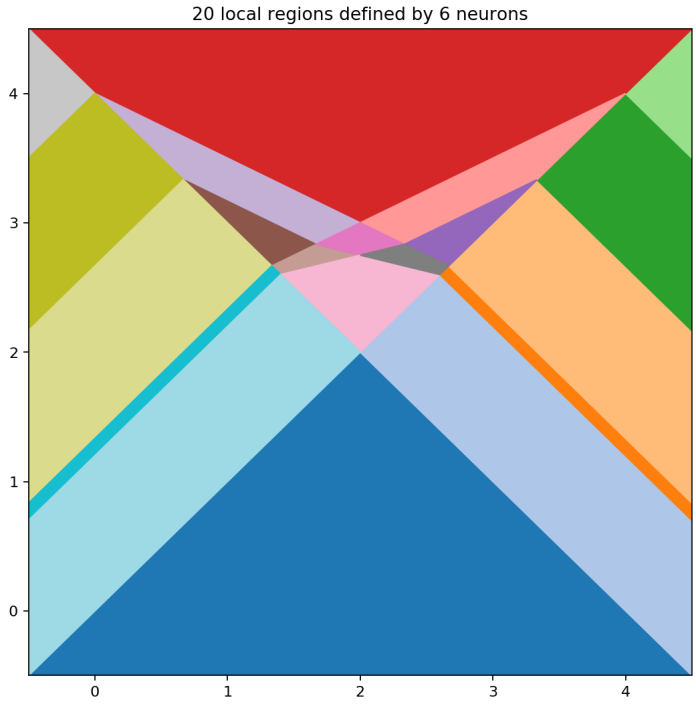


Figure 3.2: Example of a collection of bent hyperplanes, defined by a FFNN with ReLU activations and three layers with 2 neurons each. The network is the same as in Figure 2 in [74]: it has a 2 dimensional input space, and the outputs of the neurons are $h_a = max\{0, -x_1+x_2\}$, $h_b = max\{0, x_1+x_2-4\}$, $h_c = max\{0, -h_a - 3h_b + 4\}$, $h_d = max\{0, -3h_a - h_b + 4\}$, $h_e = max\{0, h_c + 3h_d - 4\}$, and $h_f = max\{0, 3h_c + h_d - 4\}$. The image has been created with my implementation of the sweep hyperplane method [30, 31, 62, 75], which is used as a function in the MBH algorithm (Algorithm 1).

change only at the hyperplanes, each region between the hyperplanes corresponds to exactly one activation pattern $\mathcal{A}$. Those regions are called *activation regions*:

Definition 3.8. Activation region $\mathcal{R}$ [31]
Let F be a rectified FFNN with k neurons, the parameters Θ, and $\mathbf{a}_{ReLU} = \{0, 1\}$ the set of possible activation statuses of the neurons of F. The activation region corresponding to an activation pattern $\mathcal{A}$ of F, is

$$\mathcal{R}(\mathcal{A}; \Theta) := \{x \in \mathbb{R}^{d_0} \mid (-1)^{a_i} f_a(x_i; w_i, b_i) > 0, \qquad i \in [0, k]\} \tag{3.4}$$

where d_0 is the dimension of the inputs space, w_i the fixed weights, b_i the fixed bias, and $a_i \in \mathbf{a}_{ReLU}$ the integer representing the activation status of neuron n_i.

It is good to note that the definition of an activation region (AR) provided above applies only to rectified networks, as used in the original source [31]. For the equation to be applicable to other PWL activation functions, such as the hard tanh, the term $(-1)^{a_i}$ should be changed to support the possible activation states of the activation function. See Definition 3.1 for how the possible neuron activation statuses depend on the activation function.

Another way to define activation regions is by leveraging the definition of hyperplanes. If one removes the hyperplanes from the input space, then the *connected components* remaining are the activation regions.

Definition 3.9. Non-empty Activation Regions [31]
Non-empty activation regions are the connected components of the complement $\mathbb{R}^{d_0} \setminus \bigcup H_F$, i.e. a set of points in the input space delimited by the hyperplane arrangement H_F (possibly open towards infinity).

This definition is widely used [26, 60, 63, 65], although it is mistakenly labeled to define *linear* regions instead of activation regions. As pointed out in [31], two adjacent regions separated by a hyperplane could coincidently have the same linear function, in which case together they would form one region in the terms of linear regions.

An important property of activation regions[1] is their convexity:

Lemma 3.1. *Activation regions are convex polytopes in the input space [31]. Let F be a rectified network. Then for every activation pattern $\mathcal{A}$ and any set of parameters Θ of F, each activation region $\mathcal{R}(\mathcal{A}; \Theta)$ is convex.*

Intuitively, the convexity of a region means that a closed segment connecting two points, which belong to the region, is always completely inside the region. For example, from 2D shapes all triangles, circles and squares are convex, but a five pointed star is not.

Lemma 3.1 is not restricted to ReLUs but holds for any piecewise linear activation function [31]. In Figure 3.2, each differently colored region is a convex activation region[2].

While individual ARs are guaranteed to be convex, unions of ARs are not. Each individual neurons defines two unions of activation regions in the input space: one where the neuron is active, and another where it is not. These unions of ARs are not necessarily convex.

A neuron with an activation constraint is a decision pattern, and therefore defines a subnetwork. As discussed in the end of Subsection 3.1.1, a subnetwork is active when the processed input results in the same decision pattern which defines the subnetwork. When the activation status of a neuron is fixed, then the part of the input space which causes the neuron to have the same activation status as the constraint, is said to activate the subnetwork defined by the neuron and its activation constraint.

[1]The convexity has been proven already earlier [65], but was said to concern *linear* regions instead of activation regions.

[2]For an image that showcases the building of activation regions by layers, see for example figure 1 in [65], some of the figures in [7], or figure 2 in [30] defined by the small DNN.

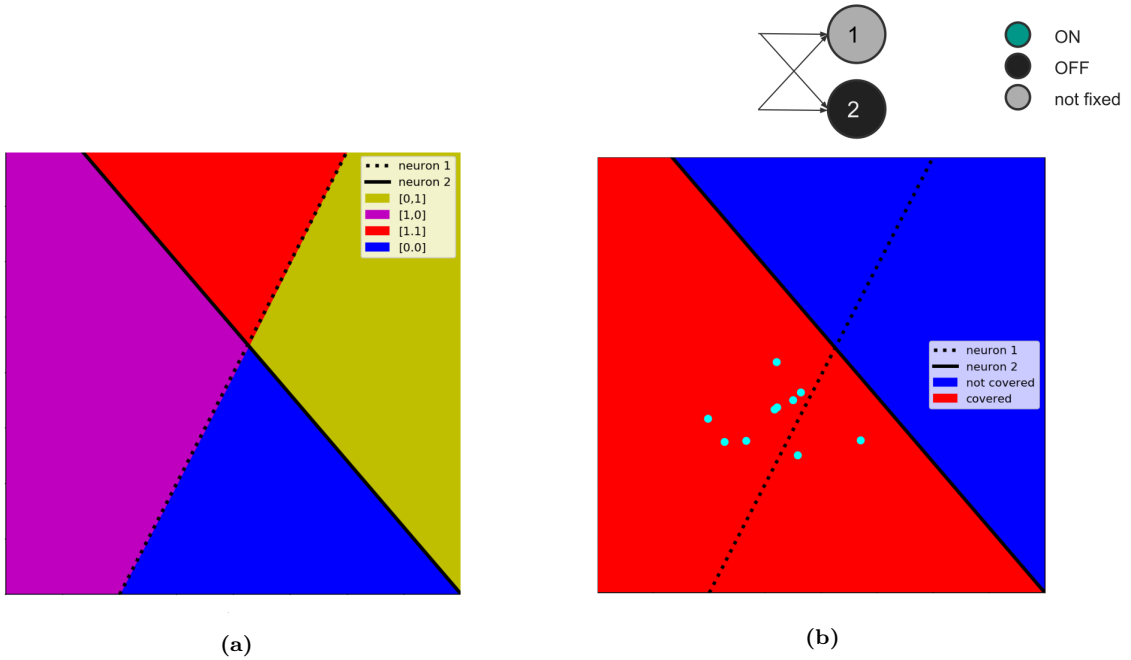


Figure 3.3: A minimal example of a union of activation regions that activate a subnetwork.
(a) A neural network with two neurons splits the input space into four activation regions.
(b) The decision pattern $\{n_2\colon 0\}$ has one activation constraint, and it defines a subnetwork. The subnetwork is activated by the union of two activation regions colored red in the subfigure (b), where the neuron 2 is off.

The concept of the input space activating a subnetwork is illustrated in Figure 3.3. The network of two neurons can have 8 different subnetworks in total. Four different subnetworks are defined by decision patterns of a single neuron ($\{n_1\colon 1\}$, $\{n_1\colon 0\}$, $\{n_2\colon 1\}$, and $\{n_2\colon 0\}$), and another four by subnetworks with two activation constraints (both neurons on, both neurons off, and another two with the neurons having different activation statuses). The latter four subnetworks correspond to the four activation regions in the figure: the decision patterns defining the subnetworks are actually activation patterns because they define the activation statuses of every neuron in the network.

The subnetworks defined by a single neuron, however, are activated by unions of activation regions. In this simple example all four possible unions defined by a subnetwork are convex, but this is not guaranteed with deeper networks that have more neurons.

A subnetwork is activated by the connected component of the input space, where the subnetwork is active. The other way around, any input that is covered by the subnetwork results in the same activations that define the subnetwork when processed by the network. More formally,

Definition 3.10. Activation of Subnetwork
Let F be a FFNN, and σ a decision pattern that defines a subnetwork S. If an input $x,\ x \in \mathbb{R}^{d_0}$, results the decision pattern σ when x is processed by F, then x activates S. Here d_0 is the dimension of the input space.

When x activates S, it can be noted with $S(x) = True$, and naturally $S(x) = False$ when x does not activate S.

The parts of the input space, which activate a subnetwork, always coincide with a union of activation regions:

Lemma 3.2. *A subnetwork is activated by a union of activation regions.*

Proof. To prove that the set of inputs which activates a subnetwork always coincides with a union of activation regions, it is sufficient to show that there does not exist an activation region with inputs that activate the subnetwork *and* inputs that do not.

Let F be a FFNN, which partitions the input space to non-empty activation regions $\mathbb{R}^{d_0} \setminus \bigcup H_F$ (Definition 3.9). Let σ be a decision pattern, which defines a subnetwork S (Definition 3.5). Let the set of inputs, which activate S, be $\mathbf{x} := \{x \in \mathcal{R}^{d_0} |\ S(x)\}$.

Now, contrary to the original assumption, I will assume instead that there exists an activation region $\mathcal{R}_x$, which includes both inputs that activate S, and inputs that do not activate S. Let the former set of inputs be $\mathbf{x}_{on} := \{x \in \mathcal{R}_x |\ S(x)\}$ and the latter $\mathbf{x}_{off} := \{x \in \mathcal{R}_x |\ \neg S(x)\}$. Based on Definition 3.10, an input activates a subnetwork if it results in the same decision pattern which defines the subnetwork. Therefore, inputs in $\mathbf{x}_{on}$ must result in a different activation patterns than inputs in $\mathbf{x}_{off}$, when processed by F. However, all inputs in $\mathbf{x}_{on}$ and $\mathbf{x}_{off}$ belong to the same activation region $\mathcal{R}_x$, and based on Definition 3.8 all inputs in the same activation region result the same activation pattern when processed by F. This is a contradiction and therefore there cannot be an activation region $\mathcal{R}_x$ which includes both inputs that activate S and inputs that do not activate S. Therefore the original assumption holds true: $\mathbf{x}$ always coincides with a union of activation regions $\forall\ S$. □

A union of activation regions that activates a subnetwork S can be called a *blanket* defined by S. The blanket *covers* the part of the input space which activates S.

Definition 3.11. Blanket B
Let F be a FFNN, and σ a decision pattern which defines a subnetwork S of F. The blanket B defined by S is the union of activation regions which activates S.

The concepts defined in this section lay the foundation for understanding the published results and use cases of activations in PWL neural networks (Section 3.2), why sparse architectures benefit from distributing the HNPs to a larger number of neurons (Section 3.3), and how the specialization of a FFNN can be measured (Chapter 4).

3.2 Related Work

In the existing literature, activations of PWL neural networks are used mainly for two purposes: to estimate the *expressivity* of the different network architectures [30, 31, 59, 60, 63, 65, 66, 74] and to *interpret* and leverage the decision process of the networks [1, 8, 15, 26, 44, 47, 56, 62, 81]. Chronologically, the former precedes the latter: the ground work on estimating the minimal upper bounds [60] dates to 2014, while papers on verifying the behaviour of PWL networks using the activations [15, 44] were published during 2017.

In this section, I will briefly review how the activations in PWL neural networks are used to 1) assess the networks' expressivity (Subsection 3.2.1) and to 2) improve their interpretability and performance (Subsection 3.2.2).

3.2.1 Activations as an Expressivity Measure

The expressivity of a neural network refers to the networks capability to express a function. The more expressive a network is, the more complex functions it can approximate.

When one considers the expressivity of a network, analyzing only the functions it is capable to approximate is a rather limited way to estimate its expressivity, since the theoretical results may assume networks of infinite width or depth [36, 60]. In addition, the functions approximated in the literature might be arbitrary compared to the real world use cases (see [60]). By estimating the number of linear/activation regions a network creates in the input space, one can compare different architectures with a more concrete measure.

Using Definition 3.2, an activation pattern can be seen as the fingerprint of a neural network's decision process given some input. Since each activation pattern corresponds to one activation region, counting activation patterns is fundamentally the same thing as counting activation regions. The former are activation states of the network for some input x, and the latter subspaces in the input space $\mathbb{R}^{d_0}$.

The number of activation regions has been widely used as an expressivity measure, and across the literature it is agreed without further scrutiny that the more activation regions a network can produce, the more expressive it is [31, 60, 59, 65, 74, 89]. The

more activation regions a network can produce, the more *non-linear* is the function the network approximates [65].

Since activation regions are defined by the complement of the hyperplane arrangement H_F, the maximum number of regions is upper bound by the number of regions a hyperplane arrangement can produce. Zaslavsky's showed in 1975 that the maximum number of regions separated by arrangement of p hyperplanes in $\mathbb{R}^{d_0}$ is $\sum_{j=0}^{d_0} \binom{p}{j}$ [87]. That theoretical upper bound has since been matched by tighter ones that leverage the knowledge on piecewise linear networks [60, 65, 74].

In [65] a theoretical and tight upper bound for the number of activation patterns (Theorem 3.1) is calculated for two different activation functions (ReLU and hard tanh) as the function of number of network layers n, the width of one layer k, and the dimensionality d_0 of the inputs in $\mathbb{R}^{d_0}$:

Theorem 3.1. Upper Bound for Number of Activation Patterns/Regions [65]
Let $F_{(n,k)}$ denote a fully connected network with n hidden layers of width k, and its inputs in $\mathbb{R}^{d_0}$. Then the number of activation patterns $\#\mathcal{A} = \#F_{A(n,k)}(\mathbb{R}^{d_0}; \Theta)$ is upper bounded by $\mathcal{O}(k^{d_0 n})$ for ReLU activations, and $\mathcal{O}((2k)^{d_0 n})$ for hard tanh.

This upper bound was later improved to a tighter one [74]. Theorem 3.2 acknowledges scenarios where there is a narrow layer inside the network: when an input dimension d_l of any layer l is small, it restricts the number of activation regions the layers after the bottleneck can create in the input space.

Theorem 3.2. Upper bound for Number of Activation Patterns/Regions [74]
Consider a deep rectifier network with L layers, n_l rectified linear units at each layer l, and an input of dimension d_0. The maximal number of regions neural network is at most

$$\sum_{(j_1,\ldots,j_L)\in J} \prod_{l=1}^{L} \binom{d_l}{j_l}$$

where $J = \{(j_l, ..., j_L) \in \mathbb{Z}^L \ : \ 0 \leqslant j_l \leqslant min\{d_0, d_1 - j_1, ..., d_{l-1} - j_{l-1}\} \ \forall l = 1, ..., L\}$. This bound is tight when $L = 1$.

The general consensus is that the deeper the network, the more activation regions it can possibly define [59, 60, 65, 66, 74]. In Theorem 3.1 the maximum number of activation regions grows exponentially with the depth of the network. Giving a more complete picture, in [74] it is shown that with a large input dimension shallow networks can have *more* activation patterns than deep ones. This is interesting for example for the image domain, where inputs are generally high dimensional.

It is highly questionable how much calculating theoretical upper bounds will contribute to the understanding of actual implementations and real life use cases. It

has been experimentally shown recently [30, 31] that deep ReLU networks stay far away from the theoretical maximum of activation regions. The findings suggest that the actual amount of activation patterns does not increase as a function of network depth, but rather depends on the numbers of neurons in the network.

3.2.2 Activations for Interpretability and Performance

Neural networks' decision process is notoriously difficult to explain [15, 26, 44, 81] and has therefore been treated as a prime example of a black box model. While the number of activation patterns (regions) does work as an expressivity measure, activation patterns themselves can serve as a tool to explain the inner reasoning of a neural network, making it more **interpretable**.

Recent tools to make DNNs human interpretable include, but are not limited to, SUMMIT [35], ActiVis [41], Network Dissection [2, 3], and Concept Activation Vectors (CAVs) [45]. All of these methods leverage the outputs of the hidden layers of the network to connect high level, human interpretable features to individual neurons [2, 3], patterns of neurons [41, 45], or both [35].

Efforts to make neural networks more interpretable by leveraging network activations include verifying network properties, e.g. providing formal guarantees of network behaviour (Reluplex [44], piece-wise linear network verification [15]), explaining network predictions (fairness certificate [81]), and understanding uncertainty in the network predictions (prior activation distributions (PADs) [56]). I will extend the work presented in [26], since it addresses all of the interpretability related topics mentioned above.

A major aspect of interpretability is to understand *why* a neural network resulted in the output $\hat{y} = F(x)$ for the given input x. I shall call the output $\hat{y}$ the *postcondition* of the input and the network (its architecture and weights) in turn can be referenced as the *precondition* of that end result. To understand why a neural network resulted in a certain postcondition, e.g. a class in a classification task, we can use activation patterns to explain the inner workings of the network.

In [26] the authors consider a more general property inference of neural networks, where activation patterns play a central role. They apply activation patterns to explain neural network behaviour, provide robustness guarantees, simplify formal proofs of model behaviour, and distill the inference process of the network. From the interpretability point of view, explaining behaviour and simplifying formal proofs of decision boundaries are really promising application domains.

The authors observe two kinds of properties using activation patterns: *input properties* and *layer patterns* (authors reference them as 'layer properties'). The former

are convex regions in the input space, and the latter as defined in Definition 3.4.

It is good to note that input properties are not activation regions, but rather unions of activation regions. They correspond to decision patterns, and are therefore a special case of blankets: while blankets in general are not guaranteed to be convex[1], input properties are blankets (Definition 3.11) and convex [26].

Activation regions characterize inputs that result in the same activation pattern when fed into the network. If two inputs, x_1 and x_2, are really close to each other in the input space, the network can process them in exactly the same manner, ending up with the same activation pattern; $AP(F(x_1)) = AP(F(x_2))$. If the two inputs were further away from each other in the input space, the resulting activation pattern would be different as well. When the inputs do share the same activation region, then the postcondition will naturally be the same:

$$AP(F(x_1)) = AP(F(x_2)) \Rightarrow F(x_1) = \hat{y_1} = \hat{y_2} = F(x_2)$$

It is good to note that this does not hold for the other direction. Having the same postcondition (e.g. the same class in a classification task), does not guarantee that the activation region, and therefore the activation pattern, would be the same:

$$\hat{y_1} = \hat{y_2} \nRightarrow AP(F(x_1)) = AP(F(x_2))$$

Activation regions are convex (Lemma 3.1). However, ARs are usually so small that very few inputs in a dataset actually share the same region: a single activation pattern may be satisfied often by only one input [26]. That is, the activation regions have usually a small *support*.

Definition 3.12. Support of a Pattern [26]
The support of a pattern, denoted by $supp(\cdot)$, is a measure of the number of inputs that follow the pattern. Formally, it is the total probability mass of inputs satisfying the pattern, under a given input distribution. In the absence of an explicit input distribution, support can be measured empirically based on a training or test dataset.

For example, if we have a dataset of images, then we can calculate the support of a decision pattern $supp(\sigma_i)$ by recording the activation patterns for every input and then counting how many inputs produced the same decision pattern σ_i.

Decision patterns (Definition 3.3) can be used to evaluate if an input is familiar or unseen [8]. However, just considering if a certain pattern has been seen can be insufficient. For example, one could be interested in the significance of patterns: which patterns appear often, and what kind of inputs cause as these patterns. The *significance* of a pattern can be evaluated by measuring its support [41, 56].

[1]See Figure 3.2 and the discussion below it.

When one tries to understand why a network resulted in some postcondition $\hat{y}$, it is not useful to observe individual inputs as a whole. It would be much more insightful to know which *features* embodied in x caused the network to decide as it did, or which inputs look semantically similar to the network.

The semantic information learned by a neural network is contained in the latent spaces between the layers, instead of individual neurons [78]. Since a FFNN processes information in a hierarchical manner, it is expected that the network operates on different abstraction levels on different layers of the network [4]. For example in the image domain, a deep convolutional neural network (CNN) learns low level features such as lines and edges on the first layers, while more complex features, such as a bike or a car, will be recognized later in the network [39, 46].

By observing patterns with respect to a certain postcondition, one can highlight which neuron activations were constant in certain decisions (postconditions) [26, 41, 56]. It is generally [1, 8, 26, 56] noted that full activation patterns are too specific to be used as the preconditions, and decision patterns are observed instead.

In [26] the layer patterns are used to explain network predictions, to prove some logical statements of form 'if precondition A then postcondition B', and to distill rules from networks. The authors use Reluplex, a satisfiability modulo theories (SMT) solver for ReLU networks [44], to iteratively relax the activation patterns to find *minimal patterns*.

A minimal pattern is a decision pattern which has the minimal amount of constraints (defined neuron activation states) so that the certain postconditions is still satisfied [26]. By solving a minimal pattern for a postcondition Y we can see which neurons are always in the same activation state when that decision is made, e.g. an input assigned to a certain class. This enables us to compare the minimal patterns between the classes and find crucial neurons or areas in the network that indicate semantically different inputs for the neuron.

To increase interpretability of ANNs, activation patterns are used for example to i) generate fairness certificates for ReLU networks to quantify biased behaviour (for example related to race or gender) [81], ii) verify a network's decisions (Reluplex [44], node phase assignments [15]), and iii) to extract semantic features that precede the final postcondition (e.g. classification) [8, 26, 56]. The information encoded in network activations can be leveraged not only to make the "reasoning" of the network more transparent, but also to make the network perform better.

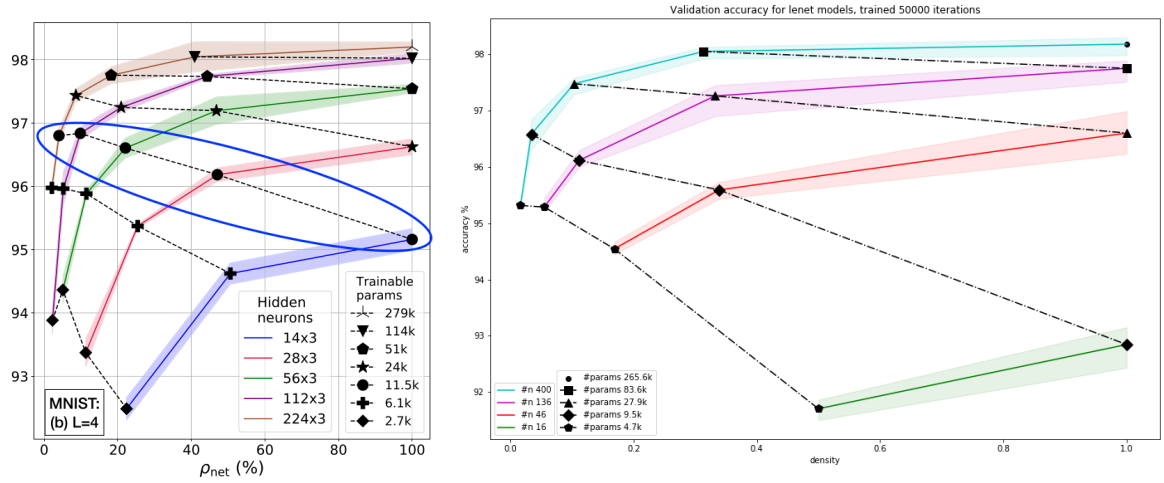
Activations of piecewise linear ANN can be used to improve the network **performance**. Existing applications include early stopping inference to reduce computation in runtime [26, 56], enhancing the networks to be more robust against adversarial attacks [1, 8, 56, 65], and network compression to save storage space and memory [47].

The work in [47] is the only source I encountered that uses network activations to achieve sparsity in neural networks and hence the first one to combine these two lines of research.

Activations have been used to estimate neural networks' expressivity, make the networks more interpretable, and to improve their performance. The former has been leveraging the activation regions in the input space and the two latter the activation patterns inside the networks. By leveraging the connection between activation regions and patterns, I will use the network activations to offer an explanation for the good performance of sparse neural networks.

3.3 Quantity over Quality with Hyperplanes

Sparse neural networks with random sparsity outperform dense models that have the same number of hidden network parameters [14, 52, 91]. This is demonstrated in Figure 3.4, where the performance of sparse and dense networks is compared in two experiments.



(a) (b)

Figure 3.4: The validation accuracy of large, sparse networks compared to small, dense networks with the same number of HNPs. All networks are trained to classify images of hand written digits from Digits MNIST -dataset. Solid lines show the accuracy of networks with a fixed size, and dashed lines connect networks with the same number of HNPs. In both images the x-axis is density, growing from 0 (all HNPs are pruned) to 1 (none of the HNPs are pruned). The y-axis is validation accuracy, with slightly different intervals between the images.
(a) Networks with 3 hidden layers, figure and experiment from [14].
(b) Lenets with 2 hidden layers from Section 6.2: each hyperparameter setup (n neurons and density d) is averaged over 5 models, confidence intervals are min and max. Each architecture has two hidden layers, which have $(3 * n/4, n/4)$ neurons, where n is the total number of hidden neurons.

In Figure 3.4 the black, dashed lines descend when the density grows. In other words, networks with the same number of HNPs, but higher density, achieve a smaller validation accuracy.

Neural networks create a collection of bent hyperplanes [31, 63] which partitions the input space, as discussed in Section 4.2. The number of the hyperplanes in the input space is the same as the number of neurons in the network. When each neuron has at least one input connection, the number of weights does not affect the number of hyperplanes the network creates. Therefore a sparse architecture, which has more neurons than a dense network with the same number of HNPs, defines more hyperplanes in the input space.

Because of this there are benefits from distributing the learnable parameters over a larger number of neurons, creating a network with more sparse neurons instead of few fully parameterized ones. While some dependencies between inputs cannot be learned with a too sparse network[1], this critical level sparsity turns out to be really big (over 90%) for many image domain problems [14, 18, 20], even though it is not rigorously defined nor exhaustively researched. In the existing literature, the term *critical sparsity* is used to describe the threshold sparsity, after which the performance of the sparse model quickly decreases. In Subfigure 3.4.a this critical sparsity is visible, as the performance of the most sparse networks declines sharply at the left edge of the subfigure.

The robustness against pruning, observed in the image domain where the input space has usually many hundreds or thousands of dimensions, suggests that it is better to have more hyperplanes with restricted orientation control (some of the weights are pruned) than less hyperplanes with complete orientation control (fully parameterized neurons). In other words, it is better to have many "low quality" hyperplanes than few "high quality" hyperplanes, where quality refers to level of orientation control of the hyperplanes.

In this section, I formulate two new hypotheses in order to explain the different performance observed in Figure 3.4. In Subsection 3.3.1, I argue that sparse networks can learn more complex functions, than dense architectures with the same number of HNPs. The other hypothesis is that networks with random sparsity can specialize more to high level features in the data. To understand this hypothesis, I will define *network specialization* in the following chapter.

[1]For example, to solve the XOR-problem with a 2 dimensional input space, the network has to consider both input dimensions simultaneously to solve the problem. A network with too sparse neurons which only consider one dimensions because the other connection is pruned away, cannot learn to solve this problem.

3.3.1 Approximating More Complex Functions

As explained in Section 3.2.1, the number of activation regions model creates in its input space is connected to model's ability to approximate complex functions. All formulas for counting or estimating the number of regions between hyperplanes presented in Subsection 3.2.1 include the number of hyperplanes or the number of neurons as a variable. Because sparse architectures contain more neurons than dense networks with the same number of HNPs, it can be expected that sparse models split the input space into more regions as well.

As seen in Figure 3.5, sparse networks do create more activation regions than dense networks with the same number of HNPs. This suggests that, by distributing the trainable parameters to a bigger number of neurons, the network can learn to approximate more complex functions than small dense networks with the same number of parameters.

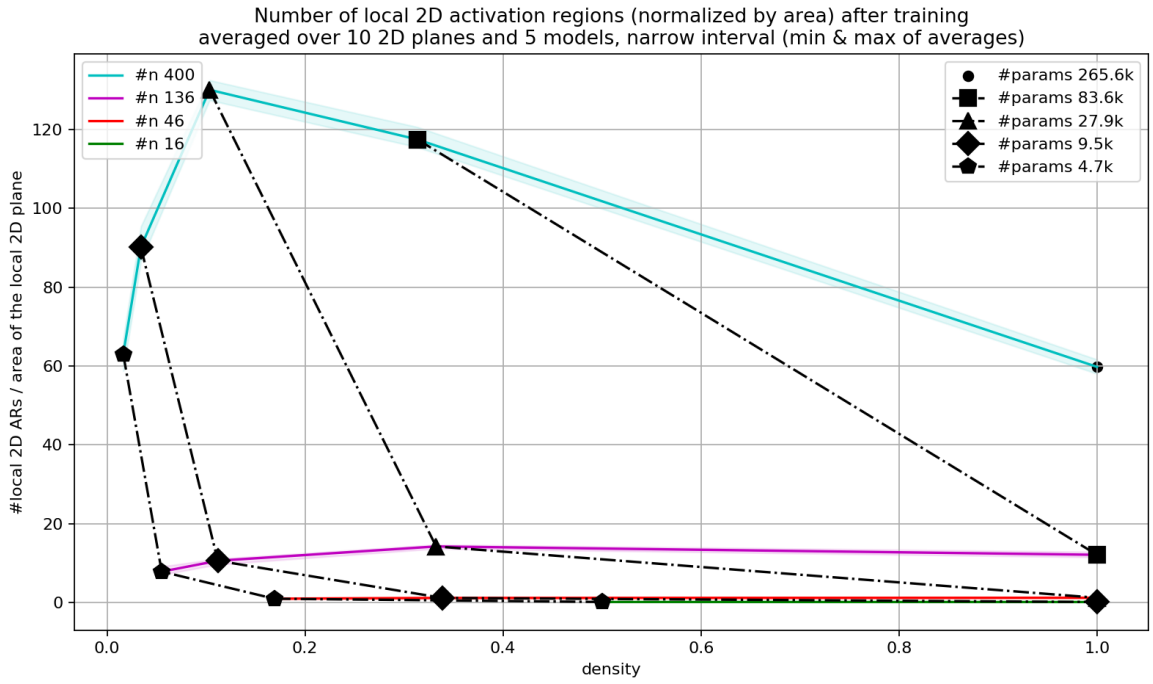


Figure 3.5: Number of local, 2 dimensional activation regions normalized with the areas of the 2 dimensional planes on which the regions were counted on, using the sweep hyperplane method described in [30, 31]. Black, dashed lines connect models that have the same number of parameters. Each hyperparameter setup (n neurons and density d) is averaged over 5 models, confidence intervals are min and max. Each architecture has two hidden layers, which have $(3 * n/4, n/4)$ neurons, where n is the total number of hidden neurons. The models are pruned randomly prior to the training. The networks have been trained on the Fashion MNIST -dataset. See Chapter 6 for more details about the experiments.

Because the sparse models in figures 3.4 and 3.5 have random sparsity, the pruning

process has not optimized the network to solve the problem as described in Section 2.3. Therefore, the ability to create more activation regions observed in Figure 3.5 is the result of the number of hyperplanes, rather than the pruning method.

The ability to approximate a more complex function with a given number of trainable parameters gives incentive to distribute parameters over larger number of neurons. This capability is not the only benefit sparse networks have over dense ones: a network with many neurons can also *specialize* more distinctly to different phenomena than a network with few, more dense neurons.

4. Network Specialization

The decision process of neural networks is not transparent [15, 26, 44, 78, 81] and they have therefore been treated as a prime example of black box models. One question which has been difficult to answer is how *specialized* a neural network is to high level features in the data.

Intuitively, a specialized neural network has learned to recognize higher level features[2] with *different* subnetworks. It is not the recognition of higher level features what makes the network specialized, but that those features are recognized with distinct structures inside the network.

In this chapter, I define *network specialization* (Definition 4.5). A specialized network can be more robust against overfitting[3], easier to interpret, and generalize better its ability to recognize different higher level features.

I start by discussing specialization on an abstract level in Section 4.1. After explaining the motivations behind specialization, I examine how it manifests in the input space (Section 4.2), and discuss how it can be measured (Section 4.3).

4.1 Philosophy of Specialization

Analyzing only the predictions of a network restricts the analysis of the performance to the last layer of the network. Calculating a validation accuracy for a network based on its predictions only tells about how well the output layer classifies the inputs transformed by the hidden layers. The classification accuracy of the networks does not tell us how the hidden layers transform the inputs, or how they partition the input space.

Classification accuracy (or other metrics calculated from the outputs of the network) can offer only indirect information about the hidden part of the network. If

[2]Here "higher level features" refer to semantic categories that have a meaning for humans. In the image domain "a shirt", "a sleeve", and "rectangular" are examples of these higher level features, in contrast to low level features like an edge or a corner. For more elaborate discussion on learned features, see [35, 88].

[3]Definitions for overfitting and generalization in [62]

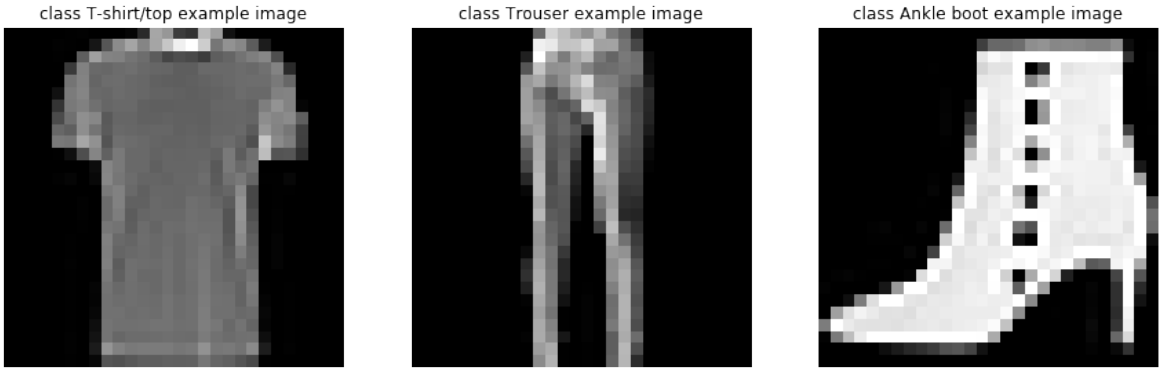


Figure 4.1: Example images from three classes of Fashion MNIST -dataset. Each image portrays a semantically different phenomenon, as humans can tell them apart from each other based on the type of fashion item.

one wants to compare hidden layers of two networks directly, they need to observe the hidden networks themselves, not the output layer. There exist many different approaches to analyze hidden layers of deep neural networks [88] and some methods that use network activations are introduced in the Section 3.2.2.

Network specialization leverages the activations in the neural network to analyze the hidden layers directly. Specialization is a property of the hidden layers of a neural network. It considers how distinctly the hidden network processes inputs that embody different semantic phenomena.

Semantic phenomena are different semantic categories humans use to classify phenomenon, such as "a bird", "a plane", "a shoe", "yellow", or "a living being". In a classification task the dataset is divided into different semantic phenomena, and a model is supposed to learn to recognize which one[1] of those phenomena each sample represents.

All classes in classification tasks are semantic phenomena, but not all semantic phenomena are labeled as classes. In Figure 4.1 there are examples from three semantic phenomena labeled as classes in the Fashion MNIST dataset. There are many other semantic categories these images can belong to. As an example, all the images portray fashion items, some of them portray objects people put their feet into, but none of them portray living beings.

If an input sample $x \in \mathbb{R}^{d_0}$ *embodies* a semantic phenomenon P, then $P(x) = True$. If x does not embody P, then $P(x) = False$.

Some semantic phenomena are *mutually exclusive*. An example of mutually exclusive semantic phenomena would be "a bird" and "a plane", assuming that an object cannot be a bird and a plane at the same time. Mutually non-exclusive semantic phe-

[1]In the case of classification task with a single target. In multi-class classification tasks networks are supposed to recognize which subset of the given semantic phenomena is present in each image.

nomena are for example "yellow" and "a bird", because an object could be a yellow bird.

If P_1 and P_2 are mutually exclusive semantic phenomena, then $mutex(P_1, P_2) = True$. If the phenomena are not mutually exclusive, then $mutex(P_1, P_2) = False$.

The recent advances in deep learning prove that NNs can learn to classify semantic phenomena from each other [46, 69]. However, just the ability to classify images with high accuracy does not tell much about the inner workings of the network. Let F_A and F_B be neural networks which classify images of fashion items with the same accuracy. Regardless of the shared accuracy, the information processing inside the networks might be drastically different from each other.

Let F_A recognize the class of an image based on the values of five pixels in the bottom left corner of the image, i.e. it exploits some artifacts in the dataset. Simultaneously F_B has learned a more general representation of the data, and is robust against value changes of individual pixels. It can be argued that F_B has learned to solve the problem "better", even though the accuracy of the two models is the same.

One way to solve a problem "better" is to learn to separate different semantic phenomena. Continuing the previous example, F_A does not separate between different semantic phenomena, it considers only few input dimensions (pixels) which do not have a meaning by themselves. On the other hand predictions made by F_B depend on higher level features such as the shape of the object. In other words, F_B predicts the class of an image based on the semantic phenomena present in the image.

A network is specialized when it has different subnetworks for recognizing different semantic phenomena. The ability to recognize different semantic phenomena is required for a network to be specialized, but it is not enough by itself. More formally:

Definition 4.1. Perfect Specialization w.r.t. Semantic Phenomenon P
Let F be a neural network. F is perfectly specialized to a semantic phenomenon P_1 iff it has a subnetwork that is perfectly specialized to P_1. Let S be a subnetwork of F. If the subnetwork S is perfectly specialized to a semantic phenomenon P_1, then

1. S is active when F processes an input that embodies P_1: $P_1(x) \Rightarrow S(x)$, and
2. S is not active when the input does embody a mutually exclusive semantic phenomenon P_2: $P_2(x) \Rightarrow \neg S(x)$,

for all inputs $x \in \mathbb{R}^{d_0}$.

For example, let F_B be able to differentiate between round objects and rectangular objects. It does this by having a subnetwork S_{B1} that outputs the "roundness" value of the object in the input image. F_B is not *specialized* to recognize either of the two semantic phenomena, round or rectangular objects, because it uses the same "roundness" value to recognize both.

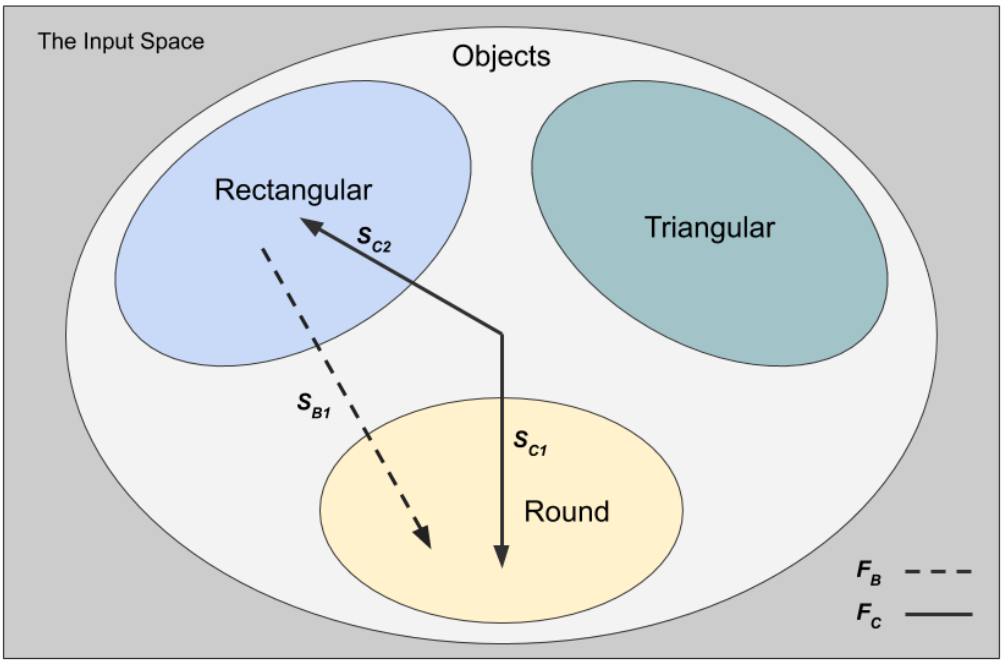


Figure 4.2: Conceptual difference between a non-specialized (F_B) and a specialized (F_C) network. Both F_B and F_C have learned to tell round from rectangular objects. Each arrow represents a subnetwork, which output grows into the direction of the arrow. F_B uses one subnetwork to differentiate between round and rectangular objects, when F_C has two subnetworks that are specialized to the semantic phenomena separately.

Let F_C be a network which solves the same task with the same accuracy as F_B. Both F_B and F_C base their predictions on the presence of different semantic phenomena in the image, and both can tell round objects from rectangular ones. However, F_C has learned two different subnetworks, S_{C1} and S_{C2}, to recognize round and rectangular objects. When F_B has one subnetwork that outputs the "roundness" of the object to decide if it is round or rectangular, F_C examines two values: one that indicates how round the object is, and another that indicates how rectangular it is. In this case F_C is specialized to recognize the semantic phenomena, the object being round or rectangular, and F_B is not.

When a network learns to recognize different semantic phenomena with different subnetworks, the outputs of the subnetworks are better aligned with the semantic categories intuitive for humans. This can make networks more interpretable.

Continuing the previous example: if the non-specialized subnetwork S_{B1} is active we cannot tell if the object in the image is rectangular or round. On the other hand, the specialized subnetworks S_{C1} and S_{C2} are active only if the object is round or rectangular, respectively. Because of this the activation statuses of the specialized subnetworks have an easily interpretable meaning, making F_C more interpretable than F_B.

If a subnetwork is specialized to certain phenomena P_a, it is not activated by inputs that embody mutually exclusive semantic phenomena P_b. For example, if a subnetwork is specialized to rectangular objects, then it is not activated by triangular objects. If a subnetwork is activated by both type of objects, then it is not specialized to either. Instead, it might be specialized to some broader semantic phenomenon, for example "shapes that have corners".

Specialization can also help the networks to generalize their ability to recognize semantic phenomena better. Let us assume that the training dataset of F_B and F_C did not contain objects with triangular shape, and the networks did not learn a subnetwork to recognize triangular objects. As illustrated in Figure 4.2, the non-specialized subnetwork S_{B1} is activated also by triangular shapes, which might get similar values as the round objects. In this case, introducing objects outside the training dataset removes the ability of F_B to reliably distinguish between different shaped objects: F_B might misinterpret triangular objects as round ones. The specialized network F_C, however, would not lose its ability to differentiate between rectangular and round objects, since neither of the subnetworks S_{C1} and S_{C2} would be activated by the triangular objects.

A specialized network recognizes semantic phenomena distinctly with specialized subnetworks. Because of this, the network can be more robust against overfitting[1], easier to interpret, and generalize better its ability to recognize different semantic phenomena. To understand better what this means in practice, I will examine how specialization manifests itself in the input space.

4.2 Specialization in the Input Space

Definition of specialization 4.1 refers to the activation status of the specialized subnetwork. As examined in Subsection 3.1.2, a network's activations can be interpreted as regions in the input space. This also applies to the activations of the subnetworks: a subnetwork partitions the input space to two (not necessarily connected) regions. In one region the subnetwork is active, and in the other it is not[2].

A subnetwork S specialized w.r.t. semantic phenomena P_1 covers the input samples that embody P_1. This follows directly from Definition 4.1 part 1 and the definition of pattern coverage 3.1.2. The second part requires that S does not cover samples that embody some mutually exclusive phenomenon P_2. In Figure 4.3 the subnetwork coverage is illustrated with an example network of two neurons.

The union of activation regions which a specialized subnetwork covers is called the

[1]Learning high level features in contrast to memorising artefacts in the data is the opposite of overfitting, see the example about F_A and F_B above.

[2]Definition 3.10 for the activation of a subnetwork.

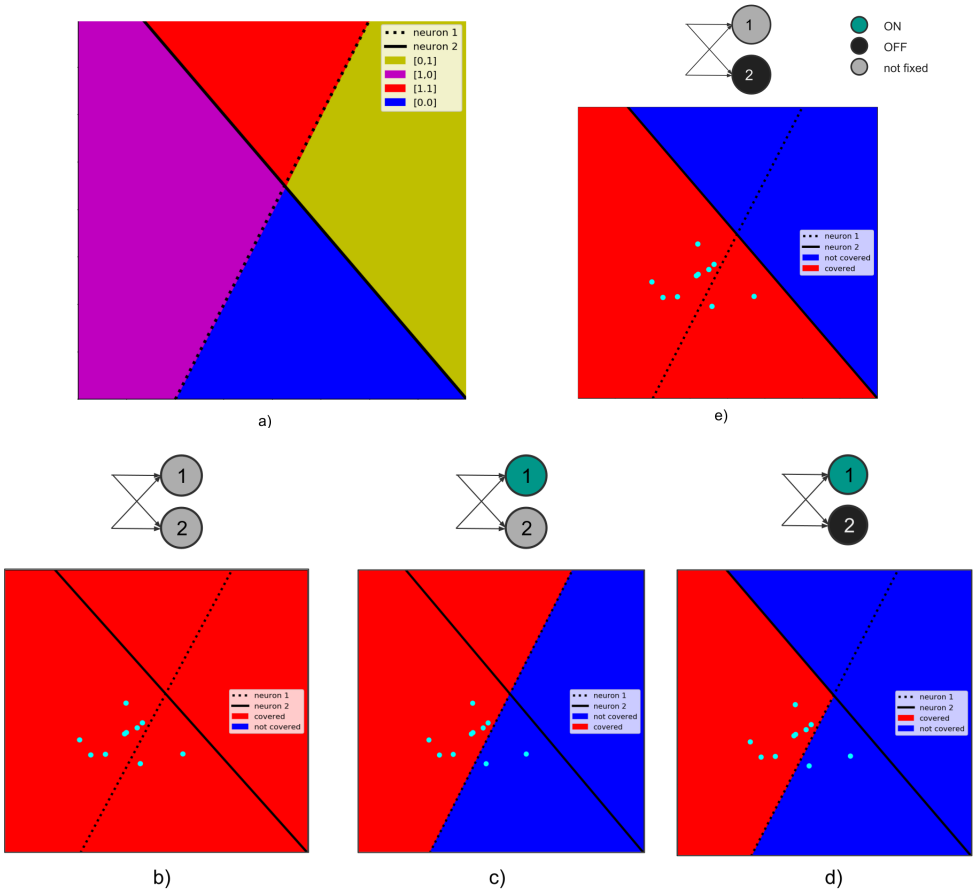


Figure 4.3: Example of subnetwork coverage in the input space. The sample network F has two neurons and a two dimensional input space. a) The activation regions spun by the network. Label "[0,1]" stands for inactive neuron 1 and active neuron 2 in the yellow region. b) The trivial blanket B_t covers the whole input space, including the 10 samples that embody some semantic phenomenon P. c) And d) illustrate subnetworks that do not cover the ten samples which embody P. e) The minimal blanket of P, defined by a subnetwork consisting of the (inactive) neuron 2.

blanket of the semantic phenomenon P (Definition 3.11). If P is a semantic category that is used as a class c in a classification task, then the blanket can be called a *class blanket* of the class c.

Not all parts of the input space do necessarily embody semantic phenomena. For example, random noise visualized as an image is definitely a point in the input space, but does not have any meaning (unless one considers noise as a semantic phenomenon). If an activation region covers such a "meaningless" part of the input space, then that region can either be or not be part of any blanket.

A semantic phenomenon P can therefore have many blankets, and these blankets can be of different size. A blanket's size is its Lebesgue measure, e.g. in the 2 dimensional example of Figure 4.3 the area of the blanket. In a three dimensional space it would be the blanket's volume. The largest possible blanket for any semantic

phenomenon is a trivial blanket, which covers the whole input space.

Definition 4.2. Trivial Subnetwork S_t and Blanket B_t
Let F be a neural network that is specialized to a semantic phenomenon P_1. Let $\mathbf{P} = \{P_i \mid mutex(P_1, P_i),\ i \in \mathbb{N}\}$ be all the mutually exclusive semantic phenomena that are present in the input space of F. If $\mathbf{P} = \varnothing$ then F has a trivial blanket B_t, defined by a trivial subnetwork S_t. S_t has no constraints on the neuron activations, and therefore B_t covers the whole input space.

The trivial subnetwork S_t can still be specialized to P_1 because there are no mutually exclusive semantic phenomenon in that input space, i.e. $\mathbf{P} = \varnothing$.

Because there can be blankets of several sizes, some blankets can be smaller than others. However, there can be only one *minimal blanket*. It is the smallest blanket which covers the semantic phenomenon P.

Definition 4.3. Minimal Blanket B_0
Let F be a neural network that is specialized to a semantic phenomenon P_1. Let $\mathbf{P} = \{P_i \mid mutex(P_1, P_i),\ i \in \mathbb{N}\}$ be all the mutually exclusive semantic phenomena. Let $\mathbf{S}$ be the set of specialized subnetworks, and $\mathbf{B}$ be the set of blankets defined by subnetworks in $\mathbf{S}$. Minimal blanket B_0 is the smallest blanket in $\mathbf{B}$, measured by the Lebesgue measure.

Because $\mathbf{S}$ includes all the subnetworks[1] that are specialized to P, the subnetwork S_0 that defines B_0 must also be in $\mathbf{S}$. Naturally it follows that B_0 is always in $\mathbf{B}$. The subnetwork S_0 is defined by the *maximal pattern* w.r.t. P.

Definition 4.4. Maximal Pattern w.r.t. a semantic phenomenon P
Let B_0 be the minimal blanket of a semantic phenomenon P, and S_0 the subnetwork that defines B_0. Maximal pattern p_0 w.r.t. P is the decision pattern that defines the subnetwork S_0.

A maximal pattern is called maximal, because adding any constraints (neurons with a defined activation status) to the pattern would invalidate the related blanket B_0. Adding constraints to p_0 would make the minimal blanket B_0 smaller in such a way that it would no longer cover the whole phenomenon P in the input space: otherwise B_0 would not be minimal. Thereby adding constraints to p_0 would make the S_0 not specialized to the semantic phenomenon P, which is not acceptable, and therefore p_0 is maximal.

[1]Here $\mathbf{S}$ is assumed to include all the appropriate subnetworks, without further requirements on the methodology how those subnetworks should be found.

In Figure 4.3 there are ten samples from one class in the input space. Two subfigures display a subnetwork defining a class blanket for that one class: b) presents a class blanket B_b that covers the whole input space and is therefore trivial, i.e. $B_b = B_t$. Subfigure e) illustrates a smaller class blanket B_e. In this case, there are no regions that could be removed from B_e by adding constraints to a neuron's activations, so B_e is the minimal (class) blanket, i.e. $B_e = B_t$.

4.2.1 Perfect Specialization Is Not Realistic

From the input space point of view, the two requirements presented in Definition 4.1 require splitting the input space into two regions: one region for input samples that embody P, and another for samples that embody some mutually exclusive phenomenon P_*. The specialized subnetwork S should be active in the former. In practice, this requirement is difficult to satisfy when solving something else than a toy example.

Consider two images of shirts from the Fashion MNIST dataset. The first is colored dark, the second one is white. Because of the different color, the images are in different corners of the input space[1] and the (Euclidean) distance between them is large compared to fashion items that have the same color but a different class.

For a subnetwork to specialize in recognizing shirts it would need to partition the input space in such a way that it would be activated only if the input would be considered as a shirt by human. Because different shirts might be far away from each other in the input space, this partitioning will probably need to be highly non-linear; it is unlikely that the shirts and all the other mutually exclusive semantic phenomena would be linearly separable in the input space.

If the subnetworks specializes to a semantic phenomenon present in classes perfectly, it would be trivial for the output layer of the network to classify all shirts correctly. Experiments show that classifying fashion items is not a trivial task (see Figure 3.4), so networks do not learn perfectly specialized subnetworks to recognize individual classes.

It is also problematic to precisely evaluate whether a subnetwork satisfies Definition 4.1. This evaluation would require labeling every single point in the input space, which naturally is not possible: how to tell the exact point when a shirt ceases to be a shirt, as the values of individual pixels are changed gradually? This is problem is visualized in Figure 4.4.

Since the perfect specialization laid out in Definition 4.1 is both unreachable and

[1]Here we assume that the input space is finite, e.g. all dimensions get values between 0 and 1, inclusive. The input space could also be open towards infinity in some or all of the dimensions, in which case talking about "corners" of the input space would not be appropriate.

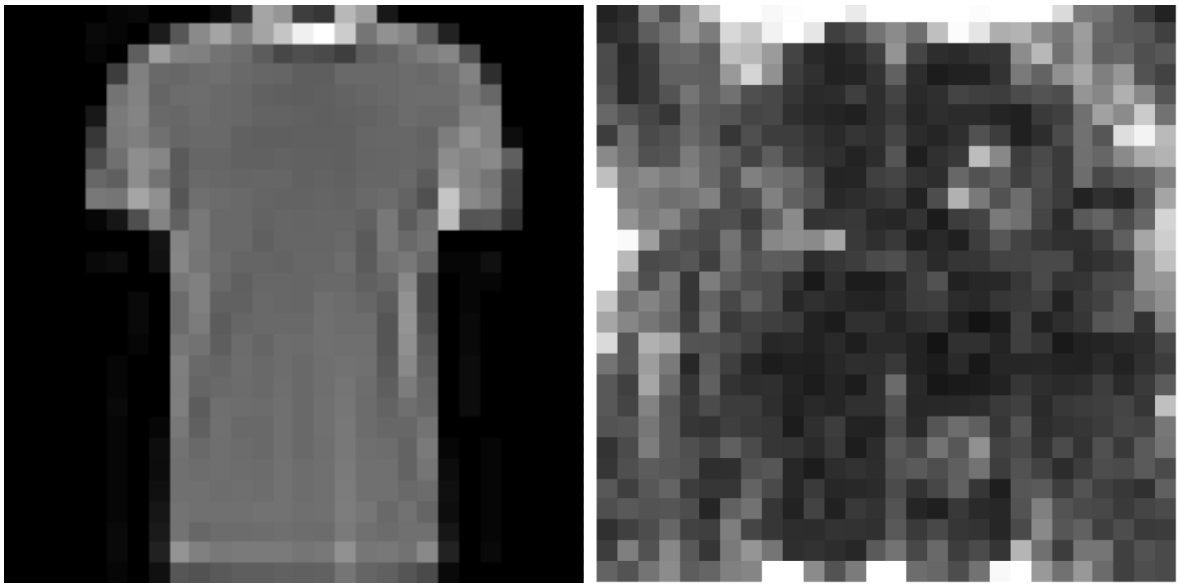


Figure 4.4: Left: an image from the Fashion MNIST dataset, labeled as T-shirt/top. Right: a point in the same input space, which does not represent a shirt. The right hand image can be obtained by gradually changing individual pixels of the left hand image. At which point does the image stop representing a T-shirt?

impossible to evaluate, we need a more practical approach to specialization. Instead of thinking of specialization as a binary state (a network has a subnetwork that either is or is not specialized to P), we can view specialization as a continuous variable.

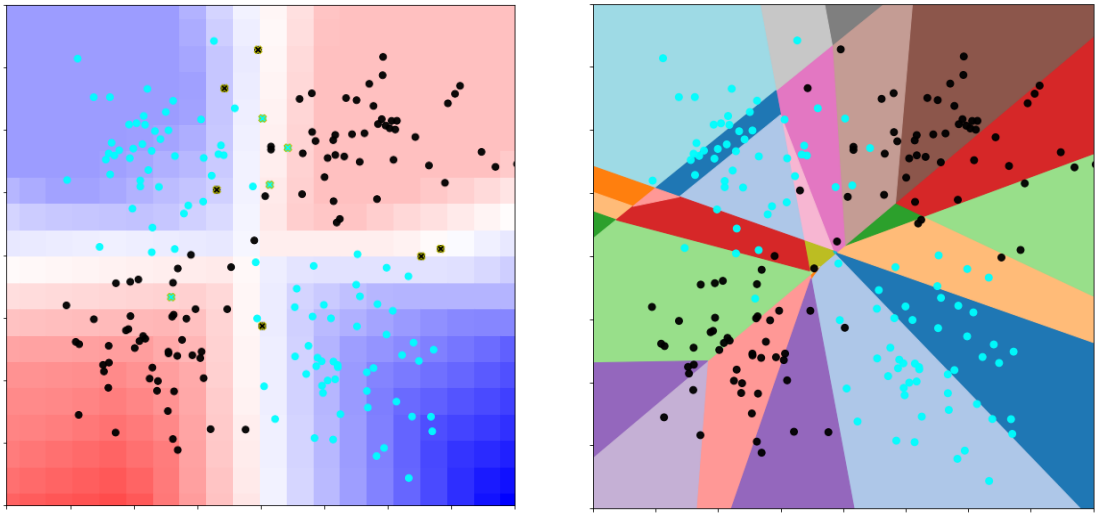
4.2.2 Specialization as a Continuous Variable

Assessing how specialized a network is w.r.t. a semantic phenomenon P requires mapping the network to a continuum from being not specialized to being perfectly specialized. In this subsection, I discuss how the two requirements of perfect specialization (Definition 4.1) should be relaxed from hard constraints to soft ones in order to make specialization a continuous variable that can be measured.

The perfect specialization (Definition 4.1) is not possible to achieve (Subsection 4.2.1), but from two non-perfectly specialized networks one can be more specialized than the other. To map a network to this specialization continuum, one can measure how well it fulfills the two criteria presented in Definition 4.1. The first criteria is arguably more important, because it considers the semantic phenomenon P we are specialized to. Fulfilling the second requirement (not being activated by mutually exclusive semantic phenomena) is secondary, because it considers other phenomena than the target of specialization.

If a subnetwork S is specialized to T-shirts, then being activated by a tank top is not that serious a shortcoming of specialization (violation of the second requirement:

tank tops are not T-shirts). On the other hand, not being activated by an image of a T-shirt (violation of the first requirement) would be clearly a shortcoming from S which is supposedly specialized in recognizing T-shirts.



(a) (b)

Figure 4.5: XOR with outliers, solved with 95% validation accuracy by a network F with (4,4,2) -neurons. (a) Decision boundary of the network's output layer after training 100k iterations. Points are the validation dataset. Crosses with a yellow border are misclassifications. (b) Collection of bent hyperplanes the hidden layers of F define in the 2 dimensional input space.

The second requirement of Definition 4.1 can be relaxed to be a soft constraint. In other words, we require a specialized subnetwork S to be activated by inputs that embody P, but we do not require S to be non-active when the network processes inputs that embody some mutually exclusive semantic phenomena P_*. In practice this can be achieved by considering no semantic phenomena to be mutually exclusive. That is, setting $mutex(P_1, P_2) = False \ \forall \ P_1, P_2$.

Relaxing the second constraint from hard to soft one removes the need to estimate the distribution of $mutex(\cdot, \cdot)$. This is convenient, since the exact distribution of $mutex(\cdot, \cdot)$ cannot be acquired because the exact distribution of phenomena cannot be acquired, as discussed in the previous subsection and illustrated in Figure 4.4.

However, a more specialized network should still be activated less by mutually exclusive phenomena, than a less specialized network. This results in the need to penalize the subnetwork's specialization score if the subnetwork is activated by *any* sample x that does not embody the phenomenon the subnetwork is supposed to be specialized to. Intuitively, if a subnetwork S is specialized to P, the less S is activated by inputs that do not embody the phenomena P, the more specialized S is. How this can be implemented in practice is discussed in Section 4.3.

Keeping **the first requirement** of Definition 4.1 as a hard constraint, while relaxing the second, would still be a too rigid criteria for a robust application of specialization. This becomes imminent when we consider outliers: datapoints that diverge from the underlying distribution and are far from the trend.

An outlier could be caused by e.g. a measurement error, or a characteristic that is not well presented in the dataset. For example, in a dataset fashion items, 99% of the shirts could be black, and 1% white. While both black and white shirts embody the semantic phenomenon "a shirt", the white shirts are outliers in that dataset.

Since white and black shirts would be far away in the input space (black shirts have small, and white shirts large pixel values), then a neural network (or any other statistical model) would not easily specialize to both black and white shirts with the same subnetwork.

In Figure 4.5 a small, dense network has learned the underlying input distribution (XOR -problem) almost perfectly (see the decision boundary in Subfigure 4.5.a). Because of noise in the data, it only achieves 95% validation accuracy. This demonstrates how it can be undesirable to (over)fit the model to the available data: fitting more closely to the data would make the network to model the underlying distribution *worse*.

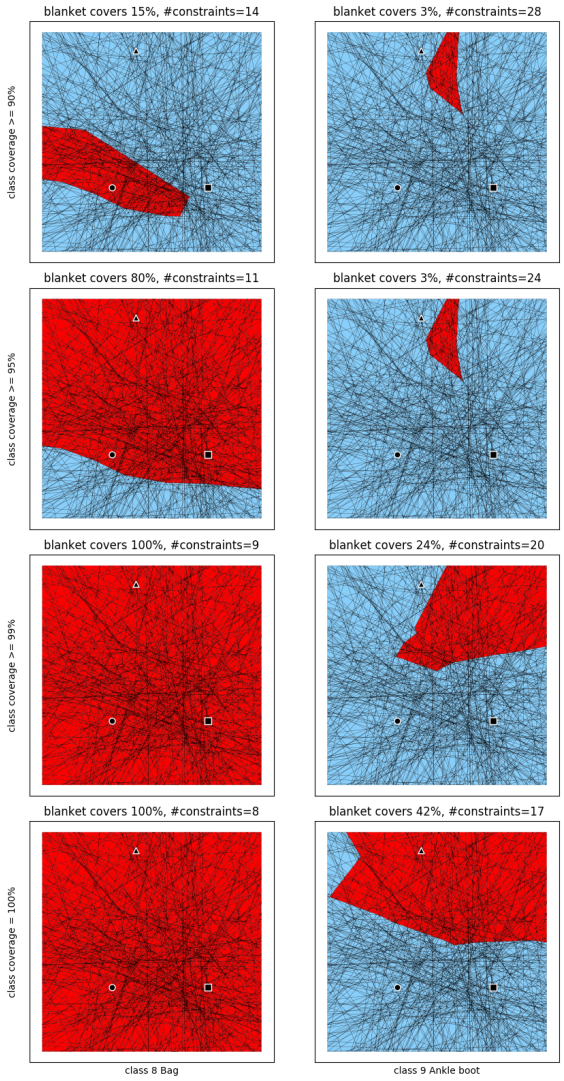


Figure 4.6: Minimal blankets to cover two different semantic phenomena (columns) with 4 different minimum coverages (rows). The phenomena are class 8 (bag) and class 9 (ankle boot), respectively. Each row has a different requirement for the minimal coverage of the phenomenon that the blanket must have. The image titles state how many percent of the 2D subspace the blankets cover, and how many activation constraints define the blanket. The network is Lenet with 400 hidden neurons, 9.5k HNPs, density 3%, and trained on Fashion MNIST. Full image in the appendix, Figure B.2.

The same applies to specialization. While the decision boundary represents how the *output layer* of the network partitions the input space, specialization considers how well the *hidden layers* of the network learn to outline different semantic phenomena in

the input space.

If the minimal blanket is required to cover 100% of the phenomenon P, then we would dismiss the existence of outliers. Instead, we can relax the first requirement of Definition 4.1 by making the required coverage of the phenomenon P to be adjustable.

In Figure 4.6 minimal blankets with different requirements for the minimum coverage (rows) are visualized for two semantic phenomena (columns). When the requirement for minimum coverage is relaxed (the first three rows) the blankets can be substantially smaller, while still covering more than 90% of the samples in the dataset that embody the phenomenon P: blankets on the first row cover $\geqslant 90\%$ of P.

Instead of always requiring 100% coverage of the semantic phenomenon P (Definition 4.1), the minimum coverage c_P can be parameterized. This way the minimum coverage of the phenomenon can be defined for each use case and dataset independently, providing robustness against outliers in the data.

Definition 4.5. Specialization w.r.t. a Semantic Phenomenon P
Let F be a feed forward neural network. F is specialized to a semantic phenomenon P iff it has a subnetwork that is specialized to P. Let S be a subnetwork of F, and c the probability that S is active when F processes an input x_P, which is randomly chosen from uniform distribution over the samples that embody P. Let $c_P \in [0, 1]$ be the minimum coverage that is required for S to be specialized to P. Iff $c \geqslant c_P$, then S is specialized to P.

The probability c can be obtained by measuring, how big a part of the subspace that embodies P also activates the subnetwork S. Let $\mathbf{x}_P$ be the set of inputs that embody P and $|\mathbf{x}_P|$ the cardinality of that set. Let $\mathbf{x}_S$ be the set of inputs that activates S. Now the probability that an input x activates S when we know that x embodies P, is

$$c = \frac{\mid \mathbf{x}_S \cap \mathbf{x}_P \mid}{\mid \mathbf{x}_P \mid} \tag{4.1}$$

For example, in the XOR problem (Figure 4.5) 50% of the (finite) input space belongs to each class. The part of the input space that is associated with class 1 is $\mid \mathbf{x}_{P_{c1}} \mid \; = 0.5 \cdot V(\mathbb{R}_{d_0})$, where $V(\cdot)$ be the function that maps a d_0 dimensional space to the corresponding hypervolume. Let S be a subnetwork that is activated by 75% of the input space, and by 90% of the inputs that belong to class 1. Now $c = \frac{\mid \mathbf{x}_S \cap \mathbf{x}_P \mid}{\mid \mathbf{x}_P \mid} = \frac{0.9 \cdot 0.5 \cdot V(\mathbb{R}_{d_0})}{0.5 \cdot V(\mathbb{R}_{d_0})} = 0.9$. In other words, the subnetwork S *covers* 90% of the phenomenon "class 1".

Since no phenomena is considered to be mutually exclusive, every feed forward network has a trivial blanket B_t (Definition 4.2) that covers the whole input space.

If a network is specialized, then it defines blankets that are smaller than the trivial blanket.

As defined on the page 43, minimal blanket B_0 is the smallest of the blankets that cover the semantic phenomenon. The blankets $\mathbf{B}$ are defined by all specialized subnetworks of the network F. Since the requirements for a subnetwork to be specialized (Definition 4.5) are more relaxed than to be perfectly specialized (Definition 4.1), the set of specialized subnetworks $\mathbf{S}$ is now formally defined as follows:

Definition 4.6. Set of specialized subnetworks S
Let F be a FFNN that is specialized to semantic phenomenon P. Let $\mathbf{x}_P$ be the set of inputs that embody P that is $\mathbf{x}_P = \{\ x \mid P(x),\ \ x \in \mathbb{R}^{d_0}\ \}$ where d_0 is the dimension of the input space. Let $\mathbf{x}_S$ be the set of inputs that activate subnetwork S that is $\mathbf{x}_S = \{\ x \mid S(x),\ \ x \in \mathbb{R}^{d_0}\ \}$. Let c_P be the minimum coverage that is required for a subnetwork to be specialized. The set of subnetworks that are specialized to P, is

$$\mathbf{S} \ = \ \{\ S \mid c_S \geqslant c_P,\ \forall\ S \in F\}$$

where c_S is the probability that S is active when F processes an input that embodies P (Equation 4.1).

The specialized subnetworks $\mathbf{S}$ define blankets $\mathbf{B}$ that cover more than c_P of the phenomenon P. In Figure 4.6 minimal blankets for the two classes of Fashion MNIST are visualized for four different minimal coverages[1] c_P. When the required coverage is smaller (upper rows), then the blanket can be smaller, since the blanket is not required to cover the whole phenomenon.

As discussed in the previous Subsection 4.2.1, evaluating if a network is perfectly specialized would require knowing the exact input distribution. In real world problems that distribution is unknown: after all, why would we need to model the input distribution with a neural network if we would already know the distribution?

However, we can approximate the input distribution empirically by observing the data. We train models to approximate the input distribution using the data, and similarly we can approximate the distributions of semantic phenomena other than the classes we are training the model to predict. Instead of labeling all points in the input space based on them embodying P or not (i.e. knowing the exact input distribution of P), we can use the dataset to see if the subnetwork is active or not with input samples that embody P.

With the empirical approach to estimate the input distribution of semantic phenomena and the relaxation of the perfect specialization (page 39) to specialization with

[1]Since in this case the semantic phenomena are classes, the minimal coverage can be called class coverage.

soft constraints (Definition 4.5), we can measure how specialized a network is w.r.t. a semantic phenomenon P.

4.3 Measuring Specialization

Network specialization considers how distinctly a FFNN processes inputs that embody different semantic phenomena. Given some phenomenon P, the most specialized subnetwork S_0 defines a minimal blanket B_0 that covers the phenomenon in the input space. The smaller the blanket is, the more specialized the network is: by outlining P to a small portion of the input space the network is activated less by inputs that do not embody P.

The distribution of P in the input space is estimated from the (validation) data. A hyperparameter c_P is used to define the portion of inputs which embody P that the minimal blanket is required to cover.

To compare the specialization of neural networks, each network needs to be mapped to a continuous specialization value. In this section, I present a method to measure the specialization based on how a network F partitions its input space with bent hyperplanes.

I first define specialization measure $s \in [0, 1[$ and examine some of its theoretical properties in Subsection 4.3.1, before proposing an algorithm to compute it in Chapter 5.

4.3.1 Specialization Measure *s*

Specialization measure s (Definition 4.7) compares the hypervolume of the minimal blanket to the hypervolume of the input space. For the input space to have a finite hypervolume it needs to be closed, i.e. not open towards infinity. To limit an infinite, d_0 dimensional real space to be finite each dimension is restricted with an lower and upper bound so that all reasonable values belong to the interval.

The lower and upper bounds can be provided by the domain, or they can be user defined. In the image domain, for example, the limits arise naturally as pixel values are usually either floating point numbers between 0 and 1 or integers between 0 and 255 [46, 49]. If the image data is normalized by each dimension, then each dimension i could have different lower and upper bounds $\mathbf{a}_i$ and $\mathbf{b}_i$, but none of the dimensions would be open towards infinity regardless of the normalization.

Definition 4.7. Specialization Measure s w.r.t. a Semantic Phenomenon P Let F be a neural network and S_0 its most specialized subnetwork to a semantic phenomenon P. Let B_0 be the minimal blanket defined by S_0 and

$$\mathbb{R}_{in} = \{\ r \mid (r > \mathbf{a}_i\ \wedge\ r < \mathbf{b}_i)\ \forall\ i \in [0..d_0], \quad r \in \mathbb{R}^{d_0}\ \}$$

the finite input space, where each dimension i gets values between $(\mathbf{a}_i, \mathbf{b}_i)$, and these limits are saved in limit vectors $\mathbf{a}$ and $\mathbf{b}$. The network's specialization is

$$s = 1 - \frac{V(B_0)}{V(\mathbb{R}_{in})}$$

, where $V(\cdot)$ be the function that maps a d_0 dimensional subspace to the corresponding hypervolume, measured by the Lebesgue measure.

The more specialized the network is, the smaller the minimal blanket B_0 becomes, and the larger s gets. In the trivial case[1] of a completely not specialized network $B_0 = B_t$ and $V(B_0) = V(B_t) = V(\mathbb{R}_{in})$, which results $s = 0$. This is the tight lower bound of s, presented in Equation 4.2.

$$\lim_{V(B_0)\to V(\mathbb{R}_{in})} s = \lim_{V(B_0)\to V(\mathbb{R}_{in})} \left(1 - \frac{V(B_0)}{V(\mathbb{R}_{in})}\right) = 1 - \lim_{V(B_0)\to V(\mathbb{R}_{in})} \frac{V(B_0)}{V(\mathbb{R}_{in})} = 0 \tag{4.2}$$

The theoretical upper bound for s is 1 (Equation 4.3).

$$\lim_{V(B_0)\to 0} s = \lim_{V(B_0)\to 0} 1 - \lim_{V(B_0)\to 0} \frac{V(B_0)}{V(\mathbb{R}_{in})} = 1 - \lim_{V(B_0)\to 0} \frac{V(B_0)}{V(\mathbb{R}_{in})} = 1 \tag{4.3}$$

The specialization measure s can be used to analyze any FFNN and semantic phenomenon:

Lemma 4.1. *The specialization measure s is defined for all networks F and semantic phenomena P.*

Proof. It is sufficient to prove that for all networks F and semantic phenomena P i) $V(\mathbb{R}_{in}) > 0$, and ii) there exists a minimal blanket B_0.

1. Hypervolume function $V(\cdot)$ gets only positive values, and $V(\mathbb{R}_{in}) > 0$ because the input space cannot be $\varnothing$.

[1] See the definition of trivial blanket on page 43

2. All networks have the trivial blanket[1], which covers the whole input space and therefore any semantic phenomenon P that can be found in the input space.

Because of 1) and 2) s is defined for all networks and semantic phenomena. □

Values of s are positive because $V(B_0) \leqslant V(\mathbb{R}_{in}) \ \forall \ B_0, P$. This follows simply from the fact that the minimal blanket B_0 cannot be larger than the input space: the largest B_0 is the trivial blanket B_t, and $V(B_t) = V(\mathbb{R}_{in})$, which results in the theoretical lower bound 0 (Equation 4.2).

While the theoretical lower bound 0 is possible to attain, the theoretical upper bound 1 is unreachable because $V(B_0)$ is strictly positive with all reasonable values of the hyperparameter c_P.

Lemma 4.2. *Minimal Blanket B_0 has always a strictly positive hypervolume, if $c_P > 0$.*

Proof. Lets assume against the Lemma 4.2, that there exists a minimal blanket B_0 for a semantic phenomenon P, for which $V(B_0) \leqslant 0$. Let $c_P > 0$, i.e., we require that the minimal blanket covers at least some part of the semantic phenomenon P. Because the hypervolume function $V(\cdot)$ gets only positive values, then it must be that $V(B_0) > 0 \ \forall \ B_0, P$, so it must be that $V(B_0) = 0$. If $V(B_0) = 0$, then B_0 cannot cover any input samples. However, because $c_P > 0$, the minimal blanket must cover at least part of the phenomenon P in the input space. Therefore B_0 cannot be a minimal blanket. This contradicts the assumption that B_0 is a minimal blanket, and therefore $c_P > 0 \ \Rightarrow \ V(B_0) > 0, \ \forall \ B_0, P$. □

Because of Lemma 4.2 the specialization measure s will never[2] reach the ultimate specialization of 1, which is the theoretical upper bound provided in Equation 4.3. In other words, $s \in [0, 1[$.

4.3.2 On the Limitations of the Specialization Measure s

The specialization measure s presented in Definition 4.7 is not the only possible approach for measuring network specialization laid out in Definition 4.1 (perfect specialization). The approach chosen in this work evaluates specialization by measuring the

[1] A trivial blanket does not have any constraints on neuron activations, see Subfigure 4.3.b .

[2] This applies to the actual minimal blanket. It is good to note that, when the minimal blanket is *approximated* (Subsection 5.2), the approximated value can go to 0, and therefore the perfect specialization can be reached with the approximation.

hypervolume of the subspace that covers enough samples which embody the semantic phenomenon in question. The reasoning is that the smaller the subspace is the more precisely the network has learned to outline the semantic phenomenon and the more specialized it is w.r.t. that phenomenon.

This approach does not acknowledge if the same subnetwork is activated by mutually exclusive phenomena. For example, if the same subnetwork that is specialized to process inputs with cats[1] is activated by inputs with dogs as well, the specialization measure s will not penalize from it. This conflicts with the idea of a perfectly specialized subnetwork, as violation of the second constraint in Definition 4.1 is not penalized for.

Another approach to measure specialization could be to measure the overlap of subnetworks that are specialized to different semantic phenomena. The hypervolume approach is preferred over measuring the violations of mutual exclusivity because the latter requires labeling all mutually exclusive semantic phenomena in the data, while for the former only the phenomenon in question needs to be labeled. Since there can be infinitely many mutually exclusive phenomena, the implementation of mutual exclusivity specialization requires labeling of the data to an extent that the existing labeled datasets do not meet by default.

In some domains the intervals for different input dimensions might have much more variance than in the image domain studied in this work. For example, telecommunication data or medical records can have dimensions with different units and vastly different scales. The effect of input dimensions having different scales cannot be directly extrapolated from the work presented here, although the experiments on normalized data (Appendix A.3) suggest that the specialization measure is robust against linear transformations applied to the data, even on the level of individual dimensions.

In this section, I defined some theoretical properties of the specialization metric s. Based on these results, the metric can be computed in practice.

[1]The inputs can be for example audio, text, or images.

5. Computing the Network Specialization

Network specialization (Definition 4.5) can be measured with specialization measure s (Definition 4.7). The existence of a metric, however, is not enough to compute it: to compute the network specialization one needs an algorithm.

In Section 5.1 I propose Minimal Blanket Hypervolume (MBH) algorithm to compute s. This is followed by discussion how the s can be approximated efficiently with the MBH in Section 5.2. The Pytorch [64] implementation of the algorithms presented in this chapter, along with other necessary source code to use them, can be found in https://github.com/vihaton/MBH.

5.1 Minimal Blanket Hypervolume Algorithm

The abstract level algorithm to compute the specialization measure s goes as follows.

Algorithm 1: MBH algorithm

Result: Specialization measure s

F, P, c_P = a FFNN, a semantic phenomenon, minimum coverage

R, D = the finite input space, data

ars = partition_input_space(F, R) `// Activation Regions`

mp = maximal_pattern(F, P, c_P, D) `// mp defines the subnetwork` S_0

B_0 = minimal_blanket(ars, mp)

v_{B_0} = hypervolume(B_0) `// hypervolume` $V(B_0)$

v_R = hypervolume(R) `// hypervolume` $V(\mathbb{R}_{in})$

$s = 1 - \frac{v_{B_0}}{v_R}$

To compute the specialization measure s for a neural network F w.r.t. a semantic phenomenon P, we need the hypervolume of the minimal blanket B_0 and the hypervolume of the input space[2]. The former is a union of activation regions in the input

[2]Definition of s on page 51

space. To compute the hypervolume of that union we require i) the activation regions that F creates in the input space, and ii) a mapping which regions are part of B_0.

The collection of bent hyperplanes[1] partitions the input space into activation regions, and therefore line 3 solves the requirement i).

The minimal blanket is defined by the most specialized subnetwork of F. The most specialized subnetwork S_0 is defined by maximal pattern (Definition 4.4). Since the exact input distribution of P is unknown, the data D is used to approximate the input distribution and empirically solve the maximal pattern w.r.t. P (line 4). The minimum coverage c_P is used to ensure that the most specialized subnetwork S_0, which is defined by the maximal pattern, is activated by enough inputs that embody P.

With the maximal pattern we can identify the activation regions that belong to B_0 (requirement ii). On line 5 we take the activation regions that support[2] the maximal pattern. This means that all inputs which come from these activation regions induce activation patterns that satisfy the constraints in the maximal pattern. In other words, these activation regions activate the subnetwork S_0, and therefore together they form the minimal blanket B_0.

After constructing the minimal blanket computing the specialization measure is simply computing the hypervolumes of B_0 (line 6) and the input space (line 7). Finally, we use the formula for s from Definition 4.7 to compute the specialization measure of the network F.

To understand better how the input space is partitioned and how the maximal pattern is defined, I will now discuss lines 3 and 4 of the MBH algorithm in more detail in subsections 5.1.1 and 5.1.2, respectively. The first thing to note about the computations needed to partition the input space and to find the maximal pattern is that they do not depend on each other. Therefore the order of computation does not matter, and they could be parallelized.

5.1.1 Partition the Input Space

To partition the input space I use the sweep hyperplane method [30, 31, 62, 75] to construct the collection of bent hyperplanes. In the beginning there is one region, the empty input space. The idea is to add neurons one by one, and to record how the hyperplane defined by the new neuron splits the existing activation regions. After going through all the neurons, the input space has been partitioned into activation regions, that is, the distributed partitioning of the input space has been created. There is no sampling involved, so the method is exact and indeed captures all the regions created

[1] Definition of *collection of bent hyperplanes* on page 23

[2] Definition of *support* on page 31

by the network.

Algorithm 2: Partition the Input Space

```
   Result: Activation Regions ars
1  F, R = a FFNN, the finite input space
2  ars = [ R ]
3  pls = [ ]                                          // previous layers
4  for each layer l in F do
5      for each neuron n in l do
6          w, b = n.weight, n.bias
7          new_ars = [ ]
8          for each region reg in ars do
9              new_ars.append(split_region(reg, w, b, pls))
10         ars = new_ars
11     pls.append(l)
```

The Algorithm 2 is a pseudocode for partitioning a finite input space. The input space could be partitioned even though it would not be finite. However, to calculate a finite hypervolume the input space needs to be finite as well, so in this application only finite input spaces are considered.

As discussed in Subsection 4.3.1, an infinite real space $\mathcal{R}^n$ can be restricted to be finite with limit vectors $\mathbf{a}$ and $\mathbf{b}$ (Definition 4.7). In practice these limit vectors can sometimes be defined by the domain: in image domain, for example, pixel values are often limited to floating point numbers between 0 and 1 [46], and therefore $\mathbf{a} = \{ 0 \}^n$ and $\mathbf{b} = \{ 1 \}^n$.

Each activation region in ars is defined by a set of hyperplanes in the input space. These hyperplanes are defined by the neurons of the network. The algorithm proceeds through the network, splitting all the existing regions with a new hyperplane, which is defined by the neuron's weights, bias, and activation function. In the case of ReLU, the activation status of a neuron changes when the pre-activation changes sign (Definition 3.1), and therefore the hyperplane is the set of points that solve $f_a(x; w, b) = 0$ (Definition 3.6).

If a region has points from both sides of the hyperplane, then the hyperplane splits the region, and two new regions are added to the list on line 9. If the region was not split, then it is kept unchanged.

The list of layers preceding the current neuron, pls, is required to transfer the region from the input space to the latent space between layers deeper in the network. For example, if a neuron n is on the third layer of the network, each region reg needs

to be transferred to the input space of the third layer to see if n splits the reg or not. This can be achieved by applying the first two layers of the network to the points that define reg.

When the input space has been partitioned, the MBH algorithm (Algorithm 1) needs to solve which regions are part of the minimal blanket to compute the specialization.

5.1.2 Finding the Maximal Pattern

The minimal blanket is defined by the most specialized subnetwork S_0 which is defined by the maximal pattern (Definition 4.4). To find the maximal pattern is to find the decision pattern with the largest number of constraints on neuron activations, so that the blanket defined by the subnetwork still covers more of the semantic phenomenon P than the minimum coverage c_P. Because the distribution of P is inferred from the data, the minimal blanket is not guaranteed to cover P precisely, but it is an approximation by nature.

If the subnetworks are required to cover all samples that embody P ($c_P = 1$) then the minimal blanket is the intersection of blankets $\mathbf{B}$. The proof is provided in Appendix B.1. However, if the minimal coverage is smaller than one the intersection of blankets is not guaranteed to either 1) be a blanket, nor 2) cover the phenomena more than the required c_P. Because of this, finding the minimal blanket requires a more sophisticated algorithm than simply taking the intersection of blankets, when $c_P < 1$.

However, the minimal blanket which covers 100% of the samples that embody P can be utilized when constructing the smaller blankets with $c_P < 1$. Since the samples that embody P support the constraints in the minimal blanket with $c_P = 1$, then all smaller blankets can have these constraints as well. Exploiting this, we can start searching for the maximal pattern by taking the intersection of blankets with $c_P = 1$ as the starting point.

Finding the maximal pattern with $c_P < 1$ is similar to the set cover problem, which is known to be NP-complete [43]. In the set cover problem the goal is to find the smallest collection of sub-sets that covers all the samples in the union of all the sets (the universe). Finding the maximal pattern requires identifying the activation constraints for which at least c_P proportion of the samples embodying the given semantic phenomenon are covered by the corresponding blanket.

Evaluating the effect of adding new activation constraints is computationally expensive because it requires computing the hypervolume of all the activation regions that are not included in the blanket already. A simple way to solve this problem is a greedy approach: add neuron constraints with the largest individual coverage to the

pattern one by one until the blanket does no longer cover enough inputs that embody the phenomenon.

Pseudocode for finding the maximal pattern with a greedy strategy is presented as Algorithm 3. The algorithm picks input samples in the data D that embody a semantic phenomenon P (line 3). Then the filtered data is run through a FFNN F while recording the activation patterns produced by the processing (line 4).

Instead of using the activation patterns as a whole, it is possible for the user to observe only some subset of the neurons. For example, the layer patterns of the last hidden layer are expected to encode knowledge of the highest abstraction level features learned by the network [4, 24, 48], and therefore the scope of the pattern can be limited to the last hidden layer's decision pattern. The scope could be anything, and is expected to depend on the use case. In this work, the specialized subnetworks are searched by observing the last hidden layer's layer patterns (line 5).

Algorithm 3: Find the Maximal Pattern

Result: Maximal Pattern mp

1 F, P, c_P, D = a FFNN, a semantic phenomenon, minimum coverage, data
2 mp = { } `// mapping for 'neuron index → activation status'`
3 D_P = filter(D, P) `// choose inputs that embody` P
4 aps = record_activation_patterns(F, D_P)
5 lps = extract_decision_patterns(aps) `// last h-layer's d-patterns`
6 mp = choose_constant_neurons(lps) `// this is mp for` $c_P = 1 \ldots$
7 cc = 1 `// ...so the current coverage of mp is 100%.`
8 new_mp = copy(mp)
9 **while** cc $> c_P$ **and not** all neurons belong to the mp **do**
10 mp = new_mp
11 new_mp = add_neuron_with_largest_coverage(mp, lps)
12 cc = current_coverage(new_mp, D_P)

After acquiring the decision patterns, which are produced by F processing inputs that embody P, the algorithm begins to search for the maximal pattern by adding all neurons with constant activation status to the pattern (line 6). If the minimum coverage $c_P < 1$, the greedy search is needed.

From the neurons that are not yet in the pattern, the neuron constraint with the largest coverage is added to the pattern on each iteration (line 11). If adding the new constraint did not reduce the coverage too much, then the maximal pattern is updated (line 10) and the search continues. However, if the coverage of the new blanket is too small, or all the neurons are already in the pattern, the search terminates (line 9).

This method is not guaranteed to find the smallest blanket, but it is guaranteed

to make the blanket smaller with every added constraint. The former follows from the fact that the hypervolume of the blanket, i.e. the requirement for its minimality, is not directly dependent on the number of inputs that activate the subnetwork. Therefore adding the neuron with the largest coverage might reduce the size of the blanket less than another neuron with a smaller coverage. An example to illustrate this with images can be found in the Appendix C.1.

The greedy approach is, however, guaranteed to make the blanket smaller with every added constraint. Since the minimal blanket is the union of activation regions which support the maximal pattern[1], adding a new constraint cannot bring new regions to the blanket.

Lemma 5.1. *Adding a neuron constraint to a decision pattern cannot make the related blanket larger.*

Proof. Adding a constraint makes blanket B larger, if it adds at least one non empty activation region to the blanket. Let σ be a decision pattern which defines a blanket B in the input space. All the regions that belong to B support all the activation constraints of the pattern σ. Since all regions in the input space which support the existing constraints already belong to the blanket, there is no region outside B which would support the existing constraints. Therefore adding a region to the blanket requires changing the existing constraints of σ. Because adding a new constraint does not change the existing constraints in σ, adding a new neuron to the pattern cannot make the blanket larger. □

Because the greedy algorithm adds neurons in the order of their individual coverage, the smaller blankets found by the greedy approach include the constraints present in the larger blankets. More formally, let B_1 be a minimal blanket that covers a phenomenon with minimum coverage c_1. Let B_2 be a blanket that covers the same phenomenon with smaller coverage $c_2 < c_1$. Now the constraints that define subnetwork S_1 also define S_2, because the greedy algorithm adds the constraints in the same order, and B_2 is equal to or smaller than B_1 because of $c_2 < c_1$. This guarantees that all activation regions which belong to B_2 also belong to the blanket B_1. An example can be seen in figures 4.6, B.1, and B.2: every blanket of the same phenomenon consists of regions which belong also to the blankets with larger minimum coverage.

The greedy approach is an efficient way to obtain reliably the optimal or nearly optimal solution in cases when $c_P < 1$. This emphasizes the splitting of the input space as the bottleneck of the MBH algorithm in terms of computational resources.

[1]Definition for *blanket* on page 27 and for *maximal pattern* on page 43.

As discussed in Section 3.2.1, the number of activation regions in the input space is proven to be upper bound by the product of binomials of layer input dimensions (Theorem 3.2). Furthermore, there is evidence [30, 31] that the actual number of regions is exponential w.r.t. the number of neurons in the network.

Computing the exact number of activation regions in the finite input space is an expensive operation. In [74], the authors use mixed-integer linear programming (MILP) to solve the exact number of activation regions created by networks with 22 hidden neurons. They formulate the mapping of activation regions from the input space to the output space as a set of constraints, solve the problem with MILP, and then count the integer solutions to get the exact number of activation regions. Equipped with 40 CPUs and 132GB of RAM it took 31h to compute the regions of one network which has (14,8) hidden neurons.

In other words, computing the exact number of regions is computationally prohibitive. As the number of regions depends on the number of neurons [30] and the input dimensions of the layers (Theorem 3.2), input space partitionings are not possible to be computed in a reasonable time even for small networks trained on the MNIST image datasets. The number of hidden neurons in sparse networks for MNIST is several hundreds [18, 49] and the input dimension $28 \cdot 28 = 784$. With other, more challenging datasets, both the networks and the input dimensions are larger. Because of this, the input space partitioning needs to be approximated.

5.2 Approximating the Specialization Measure

To approximate the input space partitioning, I compute the partitioning on a 2D subspace instead. This approach is used to visualize the input space partitioning [62] and to count the number of activation regions [30, 31] locally, inside the 2D subspace. By observing how a FFNN partitions a finite 2D plane in the input space, we can approximate how the higher dimensional partitioning looks like. In Figure 5.1, a fully parameterized network with 400 hidden neurons splits a 2D plane to regions before and after training.

For an intuitive conceptualization, one can think of a room full of soap bubbles. The room is a three dimensional space, filled with the convex soap bubbles. If a 2D plane, e.g. a piece of paper, is put to any part of the room, it collides with some of the bubbles. If the soap bubbles did not burst, their edges would draw regions on the paper. To split a 2D plane in the input space of a MNIST dataset is to put a piece of paper in a 784 dimensional room, and observe the rims of soap bubbles left on the paper.

The 2D plane can be partitioned using the same sweep hyperplane method as in

(a) **(b)**

Figure 5.1: A dense FFNN with (300,100) hidden neurons splits a 2D subspace (a) at initialization, and (b) after training the network for 50k iterations. The network is trained to classify images from the Fashion MNIST dataset. The 2D plane is spun by three images: a shirt, trousers, and an ankle boot marked in the images with a circle, a square, and a star, respectively.

higher dimensional spaces [75]. The activation regions on the plane correspond to the ones in the input space, and are both convex (Lemma 3.1). Similarly, the regions that belong to the minimal blanket in the input space can be told apart on the 2D plane as well. In Figure 5.1 a large dense model partitions a 2D plane before and after training of the network.

The actual hypervolume of the regions can be approximated by computing the area of the regions on the 2D plane. Individual planes might not be representative of the actual hypervolume of the regions, but the expected value of the area of the region correlates with the actual hypervolume of the region [31].

The algorithm takes the network, the data, the labeling of inputs w.r.t. P, and the finite input space as its inputs. In addition to these inputs, there are hyperparameters that need to be defined for computing MBH:

1. the dimension $d \in \mathbb{Z}^+$ of the subspace, which MBH partitions;
2. $d + 1$ points from the input space, which are used to span the subspace;
3. the minimum coverage c_P; and
4. the number of subspaces specialization should be averaged over.

There is no other limit for d than the computational resources [31]. The larger d is, the more precise approximation which MBH computes. The number of points

needed to span the subspace is defined by d, but it does not define *which* points should be chosen.

By choosing the spanning points one can introduce bias to the approximation of the input space partitioning. If the points are chosen from the data, the subspaces spun by these points are influenced by the distribution of the phenomena in the data. This can make some subspaces more probable than others, and some subspaces impossible to appear.

Even if the points are chosen randomly from all the points of the input space, the subspace can be biased. For example, if the three points that span a 2D subspace are chosen from a uniform distribution, then some parts of the subspace can be outside the finite input space. This is also affected by the implementation of MBH: the parts of subspace that fall outside of the finite input space can be discarded or included in the subspace. The former introduces a bias in form of smaller subspaces close to the edges of the input space, and the latter introduces bias by measuring hypervolumes outside the input space.

Even though choosing the spanning points without introducing any bias is a highly non trivial problem, it is not a crucial one. Since the same subspace can be used for all the models that are being compared, any bias introduced in choosing points affects all of the models, and it does not distort the comparison. It is arguably a good approach to choose the points from the data: this way the subspace is guaranteed to be in the same part of the input space as the data, and that part is of highest interest when considering the input space partitioning.

The minimum coverage c_P defines how big proportion of the samples which embody P should be covered by the minimal blanket. Having a c_P close to 1 might distort the specialization measure because this would require the network to also cover the outliers in the data. If the value of c_P is too small, then it can be argued that MBH is not measuring specialization at all.

An intuitive value for c_P depends on the phenomenon that is being considered. If P is expected to have a lot of outliers, or it is not very well defined, then using a smaller c_P can be a good decision. However, there does not exist a well defined framework for choosing the value of c_P. The experimental results described in Subsection 6.3.1 give some foundation for choosing the hyperparameter c_P.

To summarize, the MBH algorithm measures the specialization of a FFNN w.r.t. some semantic phenomenon P by measuring the hypervolume of the minimal blanket which covers at least c_P portion of samples that embody P in the input space. It can be implemented to approximate the full input space partitioning by partitioning

a subspace instead, and measuring how big a proportion of that subspace the blanket covers.

With MBH and the definition of specialization, one can evaluate the hypothesis that sparse networks can specialize to high level features better than small dense networks with the same number of hidden network parameters.

6. Experiments

In this chapter, I test my hypothesis that large sparse networks specialize more than small dense networks with the same number of hidden network parameters. I start by describing the general setup of the experiments in Section 6.1. In Section 6.2, I evaluate my hypothesis. Finally, in Section 6.3, I experiment with the MBH algorithm to evaluate the effect of different hyperparameters: the minimum coverage c_P, the number of training iterations, and the depth of the networks.

6.1 Setup

The experiment setup described in this section is designed to be compatible with previous setups in the literature. Since this work is related to both network pruning and activations literature, some compromises have been made to ensure that experiments performed here would be commensurate with sources from both fields of research.

The density and number of parameters in these experiments refer to the hidden networks of the models, if not specified otherwise. The output layers are fully parameterized. As discussed in Section 2.1, the hidden networks can be seen as feature extractors which process the data for the output layer to classify. The difficulties to interpret neural networks are related to the information processing inside the hidden networks, and the following experiments aim to shed light into that black box.

6.1.1 Networks and Datasets

Lenet [49] is a widely used feed forward network architecture. The network has three layers: two hidden ones[2] and an output layer. In the network pruning literature, Lenet-300-100, which has 400 hidden neurons, is often used as a baseline for image recognition with MNIST datasets, for example in [18, 20, 28, 29, 38, 51, 83, 90]. In the following experiments I use different sized Lenet models, all of which have the same ratio of neurons between the hidden layers. That is, for some positive integer n, the first hidden layer has $3n$, and the second has n neurons.

[2]Sometimes the first hidden layer is called *input layer*, since it is connected to the input space.

In the network activations literature, however, the networks have considerably less hidden neurons. Because the number of network activation patterns (and regions) is exponential w.r.t. the number of neurons in the network [30], the number of hidden neurons used in the experiments typically varies from 22 neurons [74] to 192 neurons [31].

In my experiments, the largest models will have 400 hidden neurons, in order to be comparable with the pruning literature. Therefore, the largest networks used here are significantly larger than the ones used in network activations literature. This poses heavy requirements for the computational resources needed to perform the experiments. Since the MBH algorithm does not compute the input space partitioning for the whole input space, but approximates it by partitioning only a local subspace, the experiments presented here can be computed in a reasonable amount of time.[1].

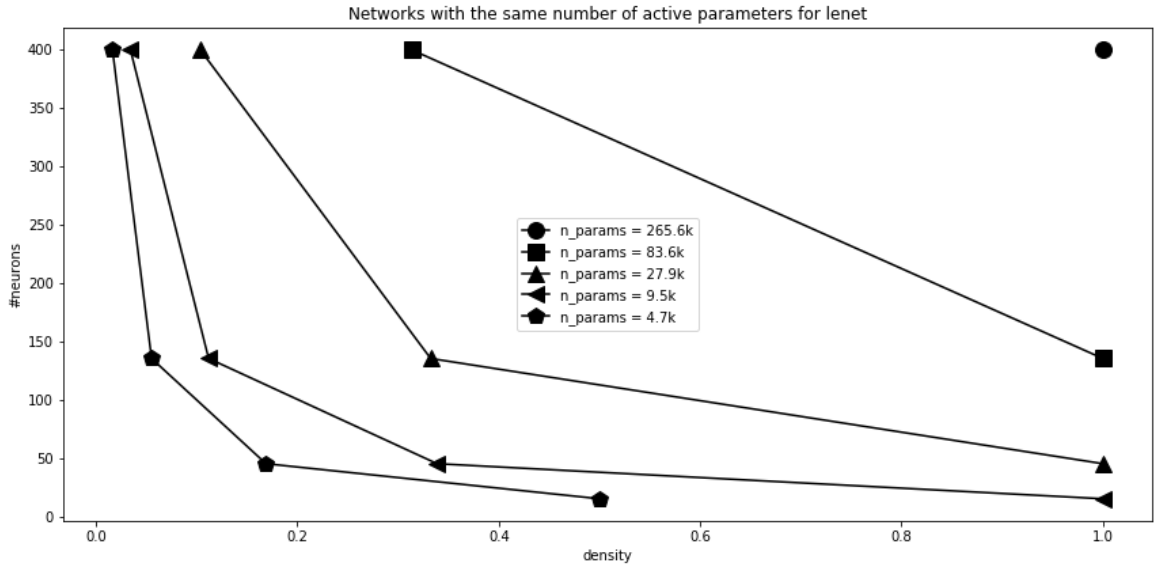


Figure 6.1: The hyperparameter setups for Lenets with 5 different numbers of hidden network parameters. The smallest and most pruned networks have 4.7k HNPs, and the largets models with 400 hidden neurons have 265.5k hidden network parameters. The number of parameters grows logarithmically, and hence the sparsity of the largest networks grows logarithmically as well.

14 Hyperparameter setups are visualized in Figure 6.1, based on the density and number of hidden neurons of the hyperparameters. The black lines connect hyperparameter setups that result in the same number of hidden network parameters. The largest networks have 400 hidden neurons and 265.5k HNPs with density 1, from which

[1]For contrast, in [74] constructing the exact input space partitioning for one network with 22 hidden neurons took 31h to compute with 40CPUs and 132GB of memory. In my work the single largest experiment contains 140 networks, from which 40 have 400 hidden neurons, and each network splits 10 different subspaces with 8 minimum coverages. The experiment took 21.6h to compute on a machine that has 16 CPUs and 60GB memory.

98.4% are pruned (density 0.016) to reduce networks' parameters to 4.7k (lowest black line, pentagons as markers).

All networks used in this work have ReLU as their activation function. ReLUs are the default non-linearity used in the network activations literature [26, 59, 63, 65, 66, 74]. In some cases it is noted that the results generalize to other piecewise linear activation functions as well: see e.g. [26] for theoretical upper bounds for hyperbolic tangent.

The datasets used in these experiments are Fashion and Digit MNIST [49]. The digit dataset has been the standard problem to showcase new theoretical results in the neural network image domain for the past 20 years [22, 29, 49], and it is still used to test new ideas in practice [18]. The fashion dataset presents a more difficult classification problem than the digit dataset. This can be seen in Figure 6.2, where networks with the same hyperparameters result in almost 10 percentage points lower top-accuracy for the fashion item classification. The experiments regarding network specialization are mostly performed on networks taught to classify images from the fashion dataset because the problem is more challenging to learn.

For more details about the network hyperparameters, refer to Appendix A.1.

6.1.2 Limitations

The experiments in this work are limited to FFNNs without any other type of layers, and both of the dataset considered here belong to the image classification domain. While there is no theoretical barrier to apply MBH and specialization to different domains and more complex network architectures which include also other type of layers, expanding the experiments further is out of scope of this work.

The semantic phenomena considered in these experiments are limited to the labeled classes in the MNIST datasets. This introduces certain bias, since the networks are incentivized to specialize to these phenomena because of the classification task. Specialization could be measured w.r.t. any phenomena that can be recognized from the data. However, for this a dataset with labeled phenomena is needed, see for example Broden (broadly and densely labeled dataset) [2].

The networks are trained for 50k iterations, without an early stopping. Because the networks do not overfit[1], they could have been trained further. However, the validation accuracy saturates after 10k training iterations, so the computational and environmental constraints motivated to limit the training iterations to 50k for all the networks.

[1] See Appendix A.2 for network validation accuracy throughout the training.

6.2 Networks with the Same Number of Hidden Network Parameters

One of the key motivations for this work is the question "why sparse networks perform better than small dense networks with the same number of hidden network parameters?". My hypothesis is that the sparse networks specialize better to high level features in the data than the dense counterparts. In this section, I evaluate that hypothesis by training over 200 neural networks to classify images of hand written digits or fashion items.

First, I will re-evaluate that sparse networks in fact do perform better than dense networks. In Subsection 6.2.1, I observe that this is indeed the case. In Subsection 6.2.2, I evaluate my hypothesis and study the relationship between specialization and validation accuracy of the networks on both datasets.

All the networks in this section are pruned at random prior to the training, so none of the benefits of optimized sparsity[1] apply to the sparse networks in this chapter.

6.2.1 Validation Accuracy

As seen in Figure 6.2, networks with smaller density (larger sparsity) reach higher validation accuracy when classifying images of hand written digits. In Subfigure 6.2.a, the dashed black lines descent from left to right as the accuracy declines, while the density grows. Even with extreme sparsity of 98.4% of the hidden network parameters removed, the largest network of 400 hidden neurons outperforms the most dense network (16 hidden neurons) by 3.61 percentage points[2].

The benefit of distributing parameters to a larger number of neurons is clearer when the number of parameters is small: with less than 10k hidden network parameters the benefit from training a sparse network instead of dense one is much greater than with 84k hidden network parameters. The benefit from distributing the learnable parameters to a larger number of weights saturates quickly after the Lenets have 100 hidden neurons (Subfigure 6.2.b).

Although classifying images of fashion items is a more difficult problem than classifying images of digits, the same trend of sparse networks outperforming dense ones

[1] See Section 2.3 for the benefits of optimized sparsity.

[2] The worst performing hyperparameter setup of the digit experiment (16 hidden neurons, 4.7k parameters, validation accuracy 91.7%) achieves 2.6 percentage points better accuracy than the best performing setup of the fashion experiment (400 hidden neurons, 265.6k parameters, validation accuracy 89.1%). Because of this, the smallest networks can be considered large enough to learn the problem, and therefore to be reasonable baselines for comparison.

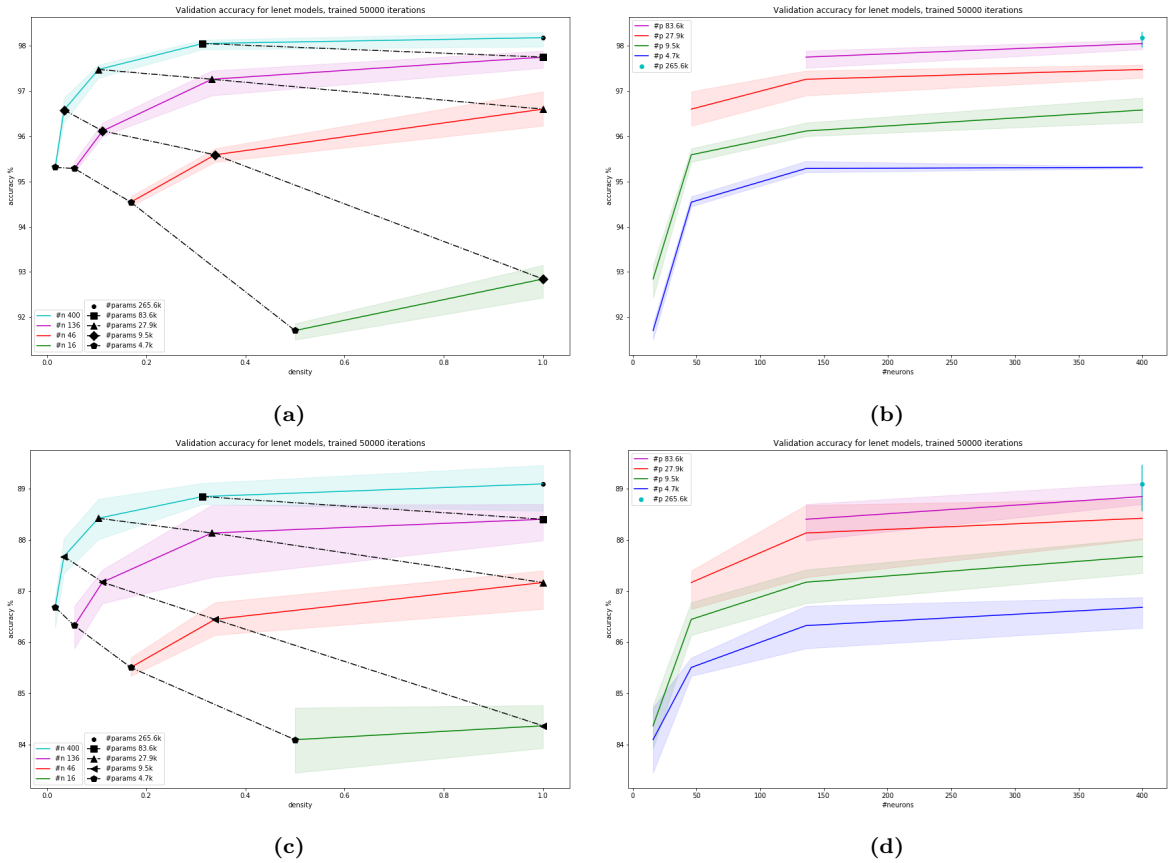


Figure 6.2: Accuracy for small dense and large sparse Lenet models: on the first row Digit, second row Fashion MNIST. In the first column the validation accuracy of trained networks is plotted over densities, on the second column the same data is plotted over the number of hidden neurons. Models for both datasets have the same hyperparameter setup, portrayed in Figure 6.1. For the Fashion dataset there are 5, and for Digit 3 models per hyperparameter setup. The intervals are min and max of the models.

is visible with Fashion MNIST as well. On the second row of Figure 6.2, the validation accuracy of larger, less dense networks is consistently bigger than of networks with a more dense structure.

The improvement in validation accuracy obtained by adding 384 hidden neurons (2.6%) is comparable to the improvement obtained by increasing the density of the largest network (2.4%). The former is a comparison between the smallest network and the largest network with the same number of HNPs, the latter between the most sparse large network (density 0.016) to the fully parameterized large network (density 1). In other words, with Fashion MNIST dataset and Lenet architecture, having 24 times more hidden neurons results in similar improvement to having 54 times more parameters in this experiment. It is good to note that the accuracy improvement is logarithmic w.r.t. *both* of the qualities, number of neurons and number of parameters.

6.2.2 Specialization

Large, sparse networks specialize more than small dense networks with the same number of hidden network parameters. In Figure 6.3 the specialization measure s is plotted for the same trained networks as in Figure 6.2. The increase in specialization saturates clearly within the experiment's network sizes: having the most sparse networks (400 hidden neurons) do not specialize significantly more than the second most sparse ones (136 hidden neurons), as seen in Subfigure 6.3.b.

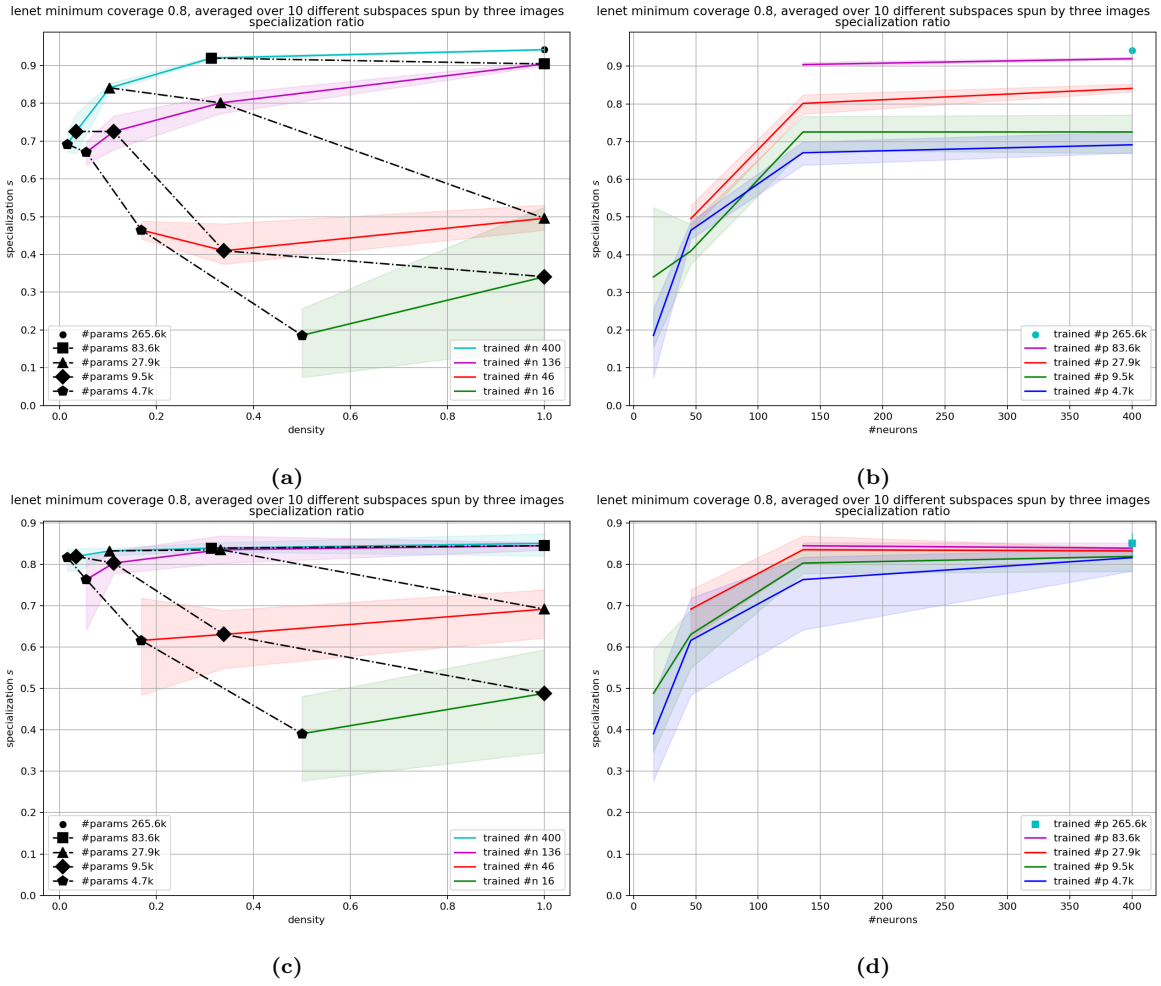


Figure 6.3: Specialization measure for small dense and large sparse networks with the same number of hidden network parameters. The models and hyperparameter setups are the same as in Figure 6.2, and the first row is Digit and the second Fashion MNIST. The same data is plotted over the network densities in the first column, and over the number of hidden neurons in the second. Minimum coverage $c_P = 0.8$.

On the second row of Figure 6.3, the specialization of networks trained on the Fashion MNIST is plotted. The same trend of saturation w.r.t. adding more neurons to the network is even more visible than with the Digit MNIST dataset in the previous figure.

As with accuracy, having more neurons in the network improves the specialization logarithmically. However, unlike validation accuracy, specialization does not clearly increase by adding more parameters to the networks (Subfigure 6.3.d). The smaller models (16, 46, and 136 hidden neurons) do specialize more when the number of parameters is increased, but the trend is much less clear than with accuracy (Subfigure 6.2.d).

In Figure 6.4, the model accuracy is plotted over the model specialization to observe the relationship between model performance and specialization more closely. There is a clear correlation between the specialization and performance of the networks.

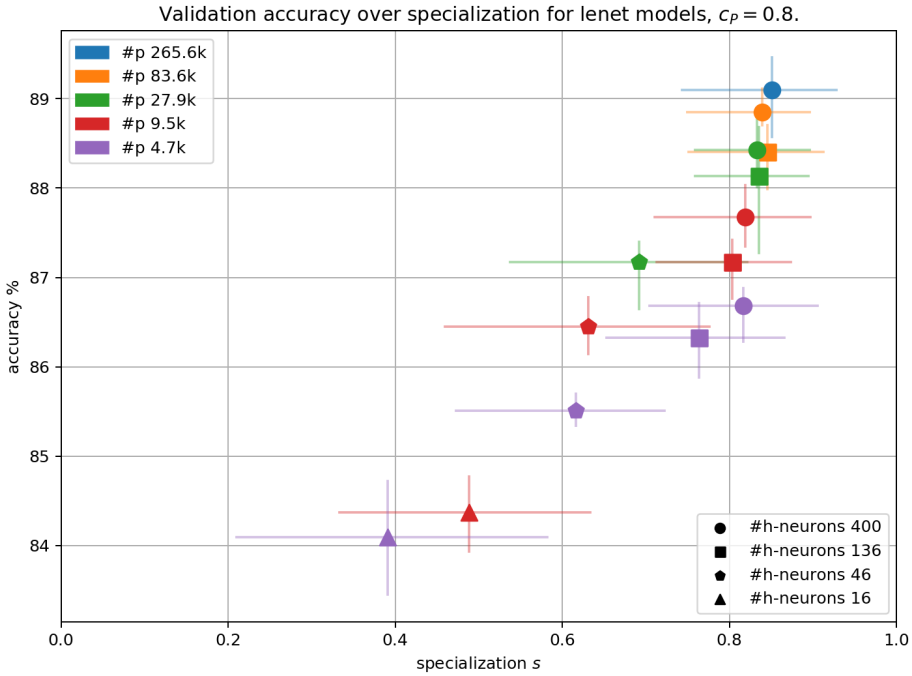


Figure 6.4: Validation accuracy of Lenet models trained on Fashion MNIST plotted as the function of specialization measure s. The minimum coverage $c_P = 0.8$. The models and the data are the same as in figures 6.2 and 6.3.

Since the accuracy can get values between [0,100] and specialization between [0,1], the increase in specialization is much steeper than in accuracy. In Figure 6.4 the smallest network with least parameters, 16 hidden neurons and 4.7k hidden network parameters, has the smallest validation accuracy and the smallest specialization. The network with 400 hidden neurons and 266k parameters has the largest accuracy and the largest specialization. The increase in accuracy, $\Delta\text{acc} = 89.096 - 84.096 = 5.0$ percentage points, is considerably smaller, than the increase in specialization, $\Delta s = (0.8511 - 0.3908) * 100 \approx 46.0$ percentage points. Therefore adding neurons and/or parameters has much larger influence on specialization than to the validation accuracy.

6.3 MBH and Hyperparameters

The MBH algorithm (page 55) measures the specialization of a FFNN by measuring the hypervolume of the minimal blanket which the network defines to cover some semantic phenomenon in the input space. As discussed in Section 5.2, the computational complexity of partitioning the input space requires an approximation. To approximate the input space partitioning with the MBH, one needs to define hyperparameters[1].

In the existing literature, where the sweep hyperplane method is used [30, 31, 62], the dimension of the partitioned subspace $d = 2$. The subspace partitioning is considerably more light weight to compute with $d = 2$ than the full input space partitioning: for example in the case of MNIST datasets, the input space dimension $d_0 = 28^2 = 784$. The two dimensional subspace, i.e. plane, is also convenient to visualize.

Spanning the subspace by using three images from three different classes is the most common choice [30, 31, 62], although this choice is not discussed or justified. In [31], some subspaces are spun by two images and the origin[2]. This fixes all the 2D planes to go through the same point in the input space. This more prone to introduce bias in the dataset, since it reduces the amount of randomness in the process of spanning the subspace: only 2/3 spanning points are chosen at random instead of 3/3.

While the method of choosing the points which span the subspace is an open research question, it is out of scope for this work. In the specialization experiments the subspace is spun with three images from the validation dataset. The images are chosen by randomizing three classes and choosing a random image from each class. This approach aims to cover the part of the input space where the data is located, while randomizing the spanning of subspace to reduce any possible bias.

In Figure 6.5, three spanning images are visualized, accompanied with three minimal blankets by three different networks. The blankets cover 100% ($c_P = 1$) of samples which embody the semantic phenomenon "ankle boot". Because of this, it is guaranteed that the black triangle marker in the upper center of the 2D subspace is covered by all the red blankets: the triangle is the ankle boot example image used to span the subspace.

The large sparse network (subfigure c) defines much more activation regions than the small dense network (subfigure b), even though both have the same number of hidden network parameters. The blanket defined by the smallest network covers the whole subspace, while the larger network (sparse and dense) cover the input space only partially. Therefore, based on this one sample subspace, the large sparse network

[1] See page 62 for further discussion on the hyperparameters.

[2] In the origin of an input space all the dimensions get the value 0.

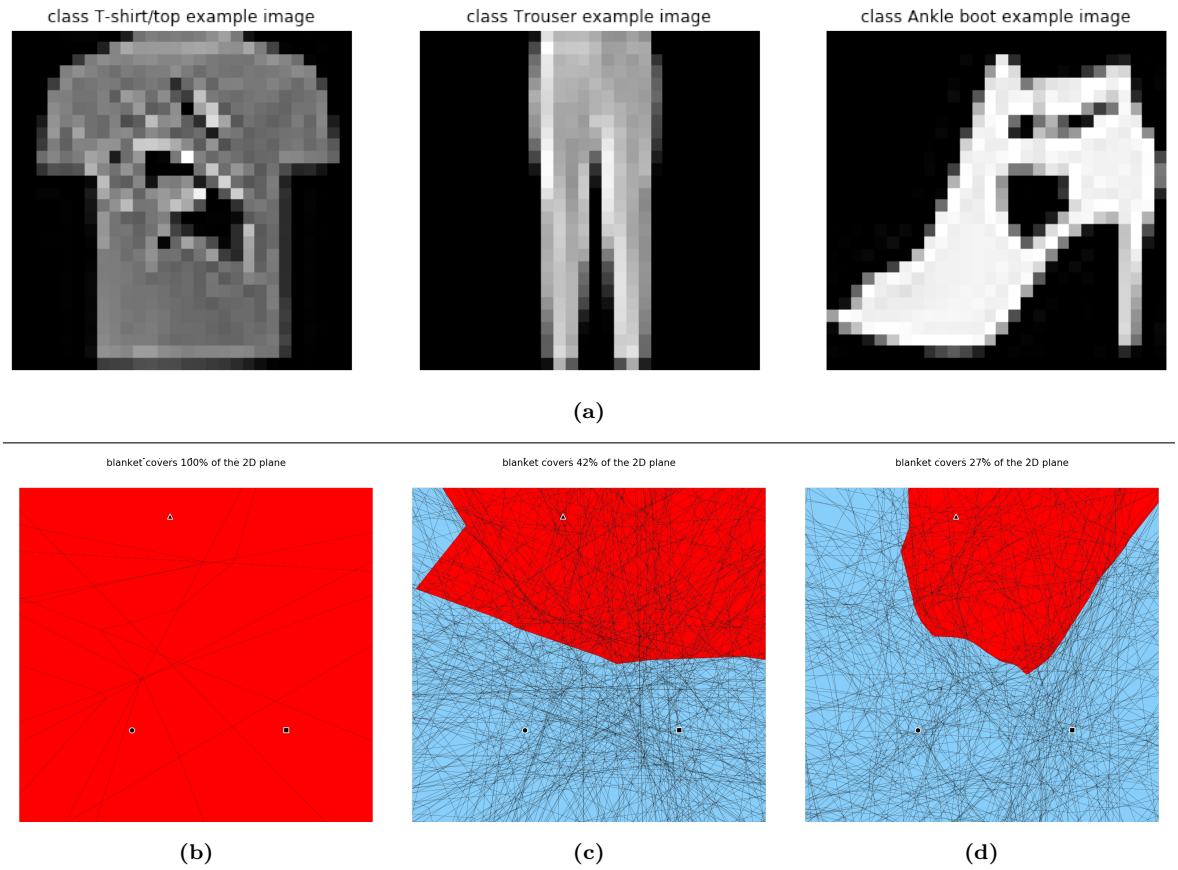


Figure 6.5: Three minimal blankets over one 2D subspace spun by three images.
(a) The three images that span the 2D subspace split by three different models in (b)-(d). In images (b)-(d) the minimal blankets with $c_P = 1$ cover the phenomenon "ankle boot" (class 9 of Fashion MNIST). The images from (a) are marked on the plane with a circle, a square, and a triangle, respectively. Since the minimum coverage is 1, the minimal blankets cover all samples of ankle boots in the dataset. Therefore the image of ankle boot from (a), marked with triangle on the plane, is guaranteed to be under the red blanket.
(b) A small dense network (4.7k HNPs, [12,4,10] neurons, density 1) covers the whole subspace.
(c) A large sparse network (4.7k HNPs, [300,100,10] neurons, density 0.03) covers 42% of the subspace.
(d) A large dense network (265.6k HNPs, [300,100,10] neurons, density 1) covers 27% of the subspace.

seems to be more specialized than the small dense one, because it defines a smaller minimal blanket. Since each 2D subspace is only a small part of the input space, the specialization measure (computed later in the qualitative experiments) is averaged over several subspaces to cancel the possible bias introduced in the approximation.

The large sparse network defines qualitatively different partitioning than the large dense network (subfigure d): the sparse network defines less *bent* hyperplanes than the large dense network. This is expected, since the pruning of weights has reduced the information flow through the network.

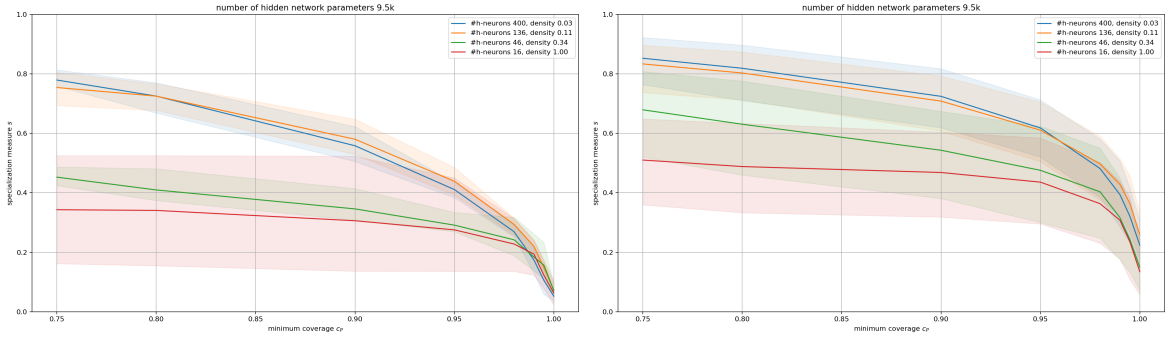
For a hyperplane to bend when it passes from a region to another, it needs to receive signal from the activation regions, and only unpruned dimensions carry signal

from an activation region through the network. Therefore, pruning connections reduces the volume of information that flows through the network, and hyperplanes defined by neurons deeper in the network cannot bend in dimensions that are pruned.[1]

In this section, I study the behaviour of specialization measured with the MBH. First, I will examine the effect of minimum coverage in Subsection 6.3.1. In Subsection 6.3.2, I present experiments to examine the development of specialization over the training process of the network, and finally I will present results from comparing Lenet models to deeper FFNNs in Subsection 6.3.3.

6.3.1 Minimum Coverage c_P

The choice of minimum coverage c_P greatly affects the specialization measured with the MBH algorithm. In Figure 6.6, the specialization of networks with 9.5k hidden network parameters is plotted over different values of c_P. With values smaller than 0.9 the specialization of all networks reduces almost linearly, when the minimum coverage is increased. However, with values larger than 0.9 the specialization measure falls quickly for all the hyperparameter setups.



(a) **(b)**

Figure 6.6: Specialization measure s over different minimum coverages for trained Lenet models. All models have 9.5k HNPs. (a) Digit MNIST, (b) Fashion MNIST.

While the specialization measure gets clearly smaller values after $c_P > 0.95$, the trend of large sparse networks being more specialized disappears as well. When the minimum coverage approaches 1, the specialization of all architectures, sparse and dense, come together, approaching 0.

[1]This is related to the bottleneck effect described in [74]: narrow intermediate layers reduce the number of activation regions the networks is able to create in the input space. If the information flow is reduced (by a narrow intermediate layer or pruning), then the hyperplanes defined by the following layers can bend less in the input space. This reduces the number of activation regions, since parallel hyperplanes do not cross, and therefore create less regions. See [75] for an introduction to hyperplane

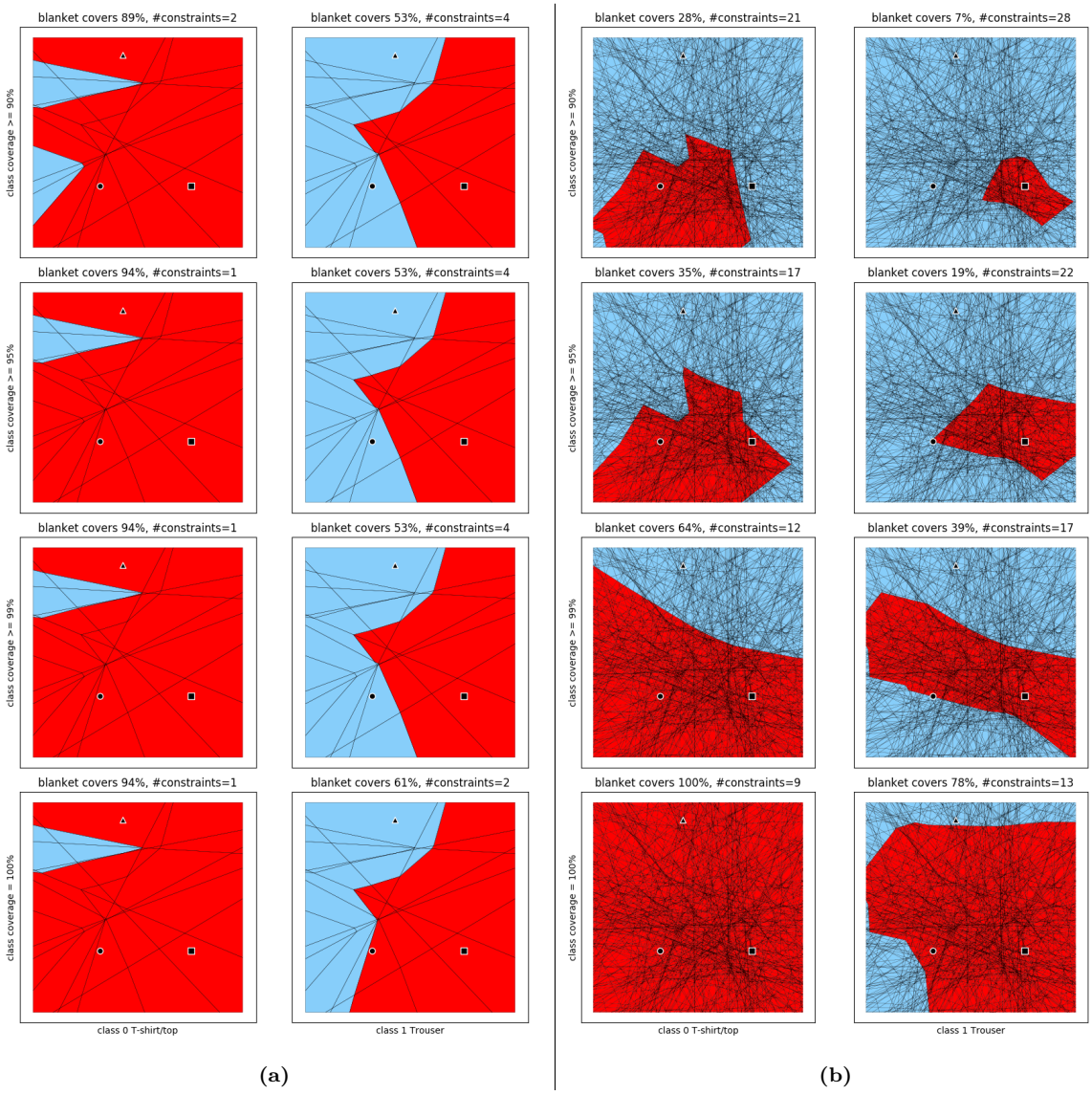


Figure 6.7: Illustration of the qualitative difference between minimal blankets defined by small dense and large sparse networks. Both models have 9472 HNPs. Each row is a different minimum coverage c_P. In both subfigures there are two columns: the first is for minimal blankets w.r.t. semantic phenomenon "T-shirt/top" and the second w.r.t. phenomenon "Trouser". Each image shows the minimal blanket with c_P defined by the row and semantic phenomenon defined by the column; the image titles tell how much of the subspace area is covered by the blanket, and how many neuron constraints define the subnetwork S_0. The subspace, over which the minimal blankets are measured, is the same in all 16 images, and the spanning images are in Subfigure 6.5.a. Full images for all the ten classes of Fashion MNIST in the appendix, figures B.1 and B.2.
(a) A small dense network with [12,4,10] neurons, density 100%.
(b) A large sparse network with [300,100,10] neurons, density 3%.

In Figure 6.7, the qualitative difference between two different networks with 9.5k hidden network parameters is visualized by presenting minimal blankets for one sub-

arrangements.

space and two different semantic phenomenon. The large sparse network, which defines over 2 orders of magnitude more hyperplanes, defines a smaller minimal blanket in 6 cases out of 8. On the bottom row, where $c_P = 1$, the small dense network defines smaller blankets than the large sparse network.

6.3.2 Specialization Over Training

The specialization measure behaves differently during training of the network with different values of minimum coverage. This is visualized in Figure 6.8, where the specialization is plotted over training with four different minimal coverages alongside the number of local activation regions. In Subfigure 6.8.b the different behaviour of minimum coverages becomes clearly visible, when plotted over the logarithmic training iterations.

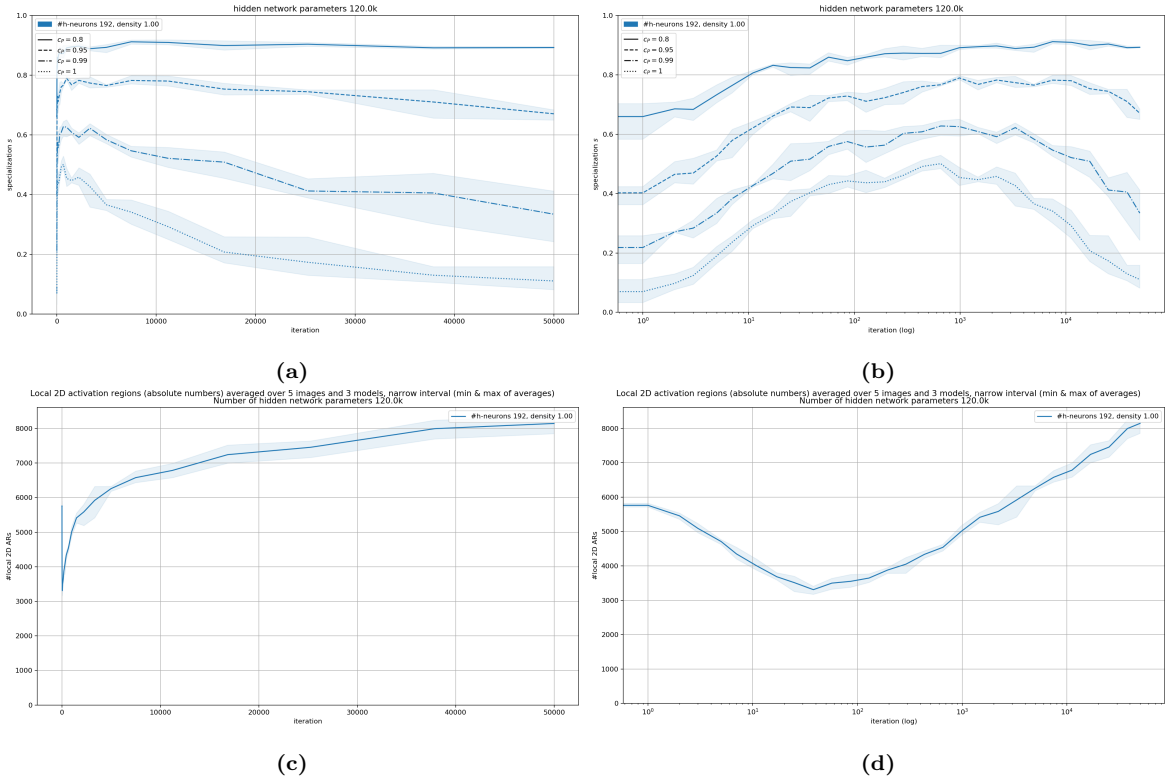


Figure 6.8: The specialization with 4 different minimum coverages (row 1) and the number of local 2D activation regions (row 2), plotted over the linear (column 1) and logarithmic (column 2) training iterations. Both columns have the same data, only the scale of the x-axis changes. Models are three Lenet models with 120k HNPs and 192 hidden neurons. Intervals are the min and max of the three models, which are averaged over 5 different subspaces before taking the min and max.

Before the training, the smaller minimum coverages result in logarithmically larger specialization. During the first hundred iterations of training the specialization increases almost linearly with all values of minimum coverage (Subfigure 6.8.c),

after which the increase saturates significantly. After a thousand iterations the difference becomes visible: specialization with larger minimum coverages starts to decline, while smaller coverages maintain their specialization or decline considerably slower.

The specialization increases over training. Before and after the training the specialization measure correlates with the number of neurons in the network. Since sparse networks have more neurons, they consequently specialize more. In Figure 6.9, the specialization of networks before and after training is plotted over the number of neurons in the networks.

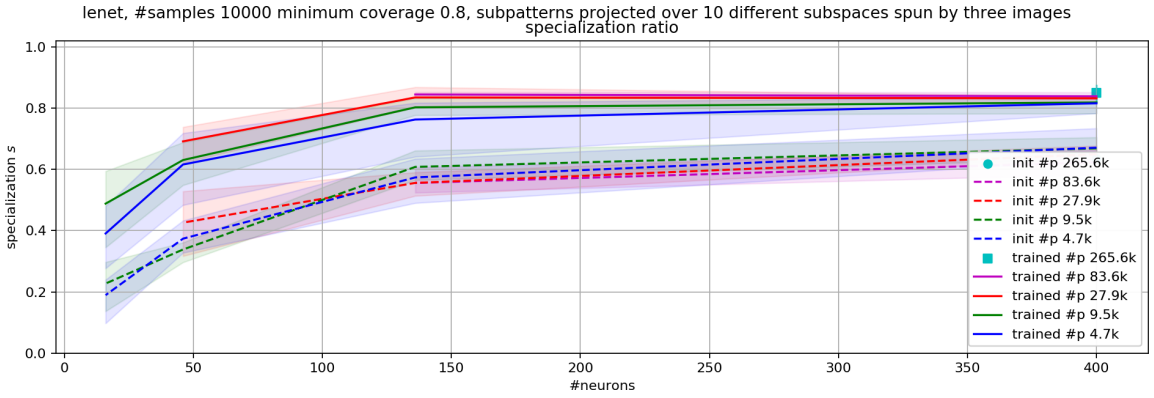


Figure 6.9: Specialization over neurons, before and after training. Lenet models trained on Fashion MNIST, minimum coverage $c_P = 0.8$.

The specialization of a network does not correlate with the number of activation regions throughout the training. In the beginning of the training the number of regions declines, and the specialization starts to grow. There seems to be a negative correlation between the number of regions and the specialization with the larger, more sparse models. However, as the training proceeds further than the first thousand iterations, the number of regions starts to grow significantly, while the specialization (with a reasonable minimum coverage) remains stable. Therefore, the number of regions does not explain the development of the specialization over the training.

In Figure 6.10, the validation accuracy, specialization, and number of regions is plotted over the training iterations for 4 hyperparameter setups with the same number of hidden network parameters. The two most sparse hyperparameter setups specialize clearly more than the two dense setups, over the whole course of training (Subfigure 6.10.c). This happens regardless of the similar accuracy (Subfigure 6.10.a) and vastly different number of activation regions (Subfigure 6.10.e).

The clear separation in network specialization w.r.t. the number of neurons suggests that there is a threshold value for which adding neurons to the network does not help it to specialize more. That threshold value seems to be between 41 and 89 hidden neurons for Fashion MNIST, in networks with two hidden layers and 11.1k

hidden network parameters (Subfigure 6.10.c).

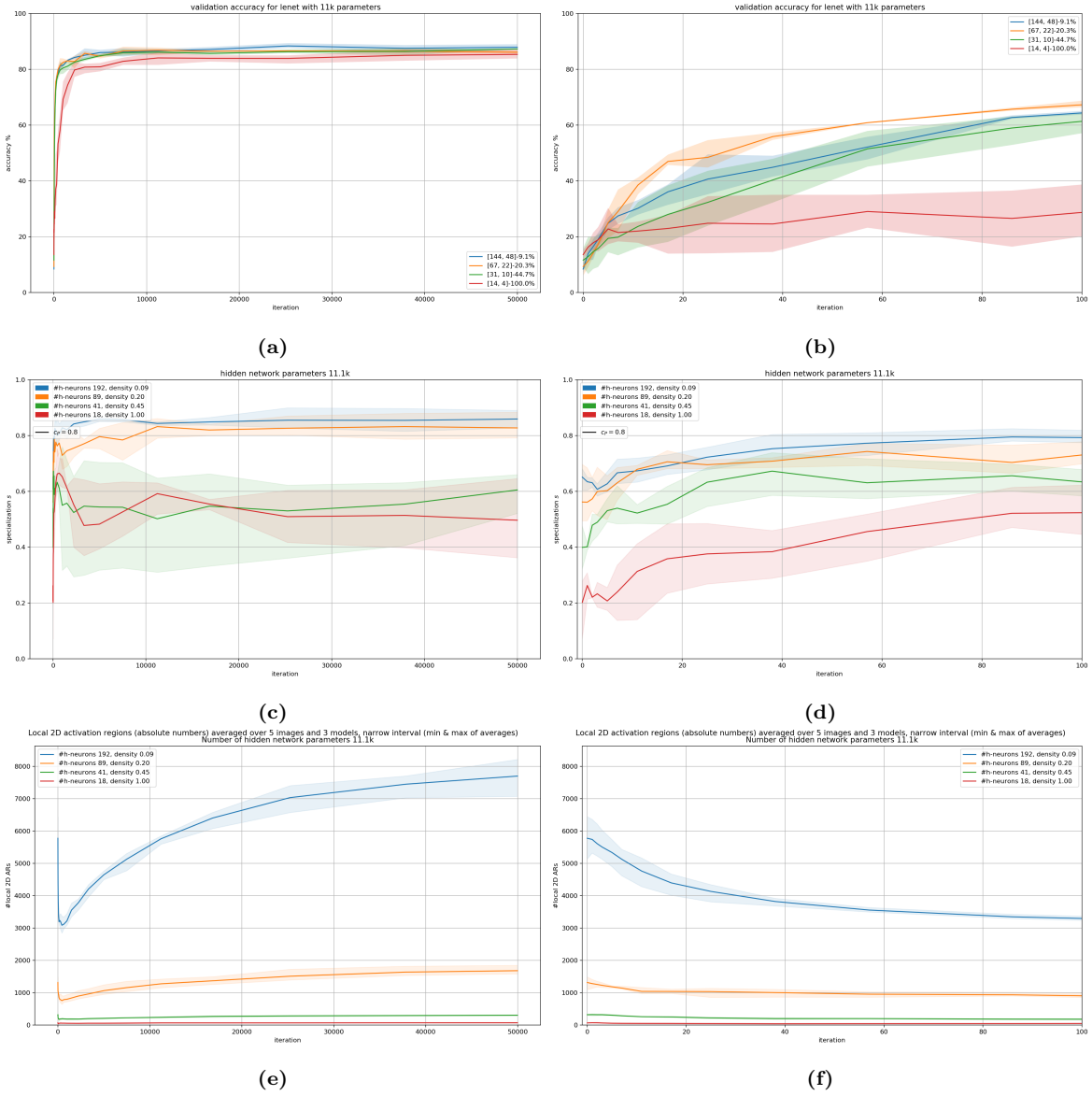


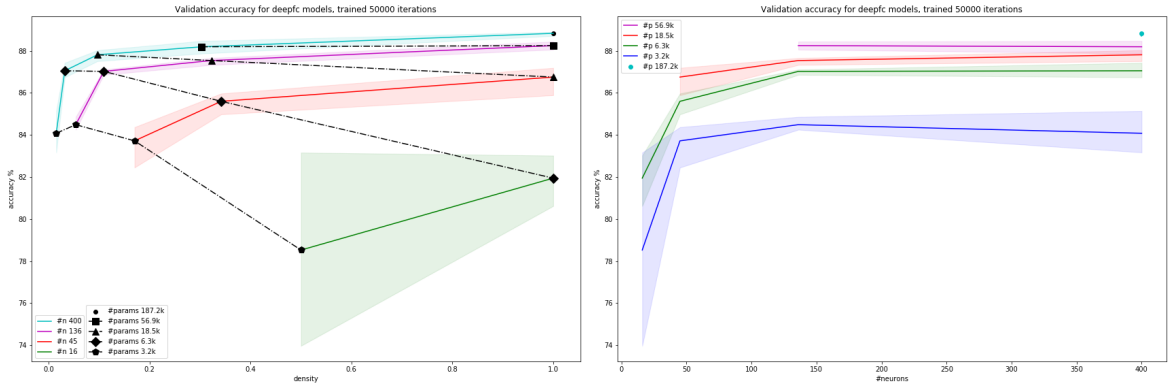
Figure 6.10: The validation accuracy (row 1), specialization with $c_P = 0.8$ (row 2), and number of local 2D activation regions (row 3) over all 50k (column 1) and the first 100 (column 2) training iterations for 4 different hyperparameter setups of Lenet models with 9.5k HNPs trained on Fashion MNIST. The number of activation regions is negatively correlated with the specialization during the first iterations of the training (the 2nd column), but the correlation does not last further into the training (the 1st column).

6.3.3 Networks with Different Numbers of Layers

As observed earlier in this chapter, the specialization measure correlates tightly with the number of neurons on the Fashion MNIST dataset[1] (figures 6.3 and 6.9). In this subsection, I study how the number of hidden layers affects the specialization of the network, when the number of hidden neurons is kept constant.

The same trends observed for the Lenet architecture can be seen with a deeper architecture which has hidden neurons distributed to three hidden layers instead of two, in proportion of $(2n, n, n),\ n \in \mathbb{N}$. In the context of this work, I will refer to the latter as DeepFC architecture. It is good to note that while the two architectures have the same number of hidden neurons, they have up to 47% difference in the number of hidden network parameters.

In Figure 6.11, the validation accuracy of DeepFCs shows the same trend of sparse networks performing better than small dense networks. The most sparse networks with 3.2k hidden network parameters seem to be beyond the critical density, since the validation accuracy is decreasing sharply between the largest models with 6.3k and 3.2k hidden network parameters.



(a) **(b)**

Figure 6.11: Accuracy for small dense and large sparse DeepFC models with the same number of HNPs, trained on Fashion MNIST. (a) Validation accuracy of 14 different hyperparameter setups over network densities, (b) the same data over the number of hidden neurons of the networks. Models have the same number of hidden neurons as Lenets in Figure 6.2, but unlike Lenets the neurons are distributed to 3 hidden layers: $(2n, n, n),\ n \in [4, 100]$.

The deeper architecture reaches a lower validation accuracy than the shallow Lenet (Figure 6.2), and there is a bigger variation between the large sparse and small dense networks. The saturation of accuracy w.r.t. the number of hidden neurons

[1] In the less challenging Digit MNIST dataset there is also a clear difference in specialization between large networks with different number of hidden network parameters (Subfigure 6.3.b). This difference cannot be seen, however, on the Fashion dataset (Subfigure 6.3.d).

happens around 100 hidden neurons, as with the Lenet models.

On the first row of Figure 6.12, the specialization of Lenet models is plotted next to the specialization of DeepFC models with the same number of hidden neurons. The trend of sparse networks specializing more is more clear with the DeepFC than the Lenet networks. Both architectures reach similar specialization with the same number of hidden neurons, regardless of the big difference in the number of trainable parameters. This suggests that the number of hyperplanes in the input space is indeed more important than the adjustability of those hyperplanes, i.e. many hyperplanes with few parameters enable a better specialization to high level features than few hyperplanes with many parameters.

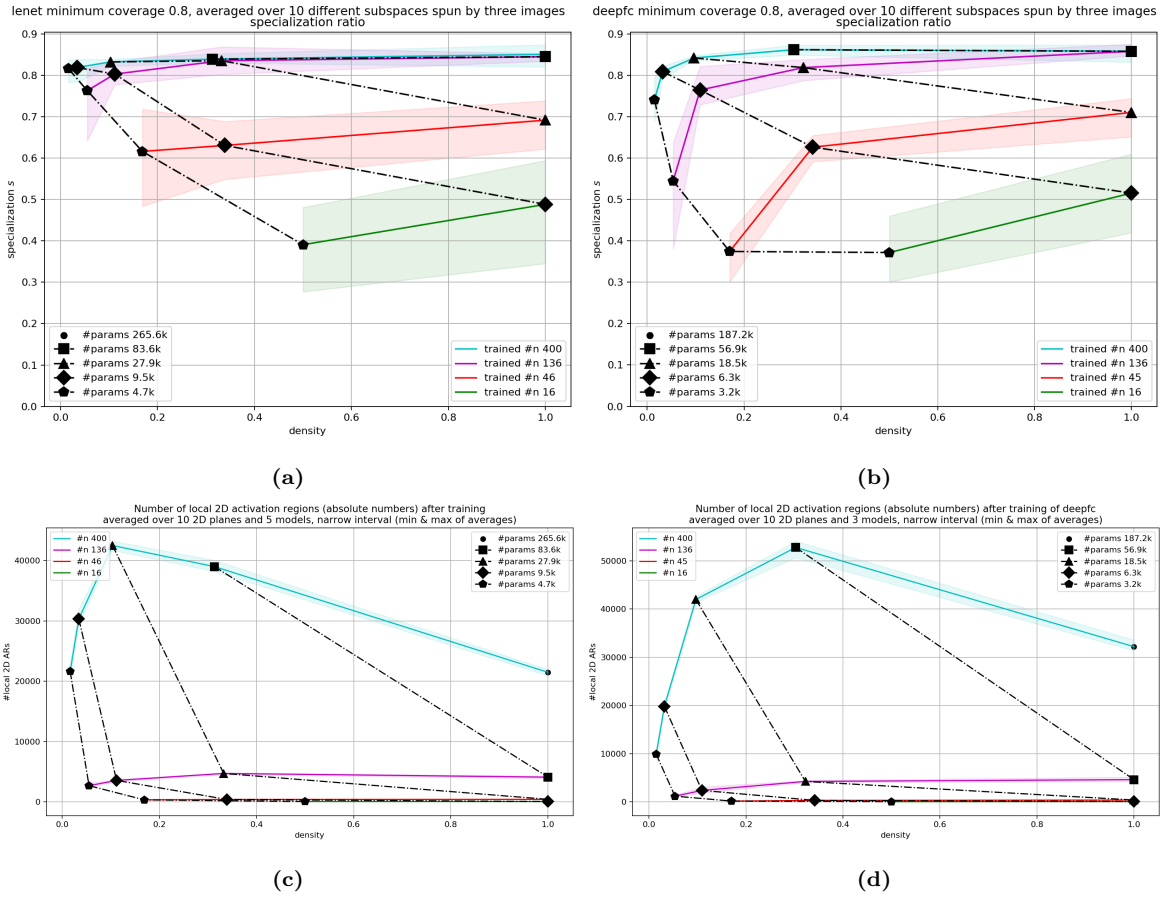


Figure 6.12: The specialization with minimum coverage $c_P = 0.8$ (row 1) and number of local 2D activation regions (row 2) of Lenets (column 1) and DeepFCs (column 2) over network densities. Models are trained on Fashion MNIST. Note the different y-axis scale between the columns. Both architectures (columns) have the same number of hidden neurons, but the DeepFC networks have considerably less HNPs.

On the second row of Figure 6.12, the number of local activation regions is plotted for Lenets and DeepFCs. While DeepFC creates more regions with the same number of hidden neurons, the numbers are relatively close to each other: the deeper architecture does not provide exponentially more activation regions. The two most pruned large

Lenets create over 50% *more* activation regions than the deeper DeepFCs. In other words, the number of hidden neurons is clearly more important than the depth of the network for the number of local activation regions.

Both architectures result in a peak in the number of activation regions created by the largest networks over different densities. The peak takes place with density smaller than 0.5 for both architectures, but for Lenet the density for which the maximum number of regions is reached is smaller than for DeepFC.

7. Discussion and Conclusions

Network specialization (Definition 4.5) is a novel concept which evaluates how distinctly a FFNN has learned to outline a high level feature in its input space. The **specialization metric** s (Definition 4.7) exploits the connection between decision patterns inside the network and the activation regions in the input space. Because of this, the specialization of FFNNs can be compared regardless of the number of hidden layers or neurons in the different network architectures: a comparison which has not been possible with the existing techniques that rely solely on the activation patterns inside the networks.

I presented **MBH algorithm** (page 55), which computes specialization of an arbitrary FFNN w.r.t. any labeled high level feature in the data. The algorithm compares the hypervolume of the subspace which the network associates with a high level feature to the hypervolume of the input space. MBH algorithm enables the activation regions to be used for making FFNN more interpretable by constructing the exact input space partitioning over a subspace instead of the whole input space, hence avoiding the computationally prohibitive construction of the full input space partitioning.

I conducted experiments to evaluate my hypothesis that **sparse networks specialize more than dense networks with the same number of hidden network parameters**. I found that this was the case in two MNIST image classification tasks (Subsection 6.2.2). This contributes to the explanation why sparse networks outperform dense models with the same number of HNPs.

I presented three other explanatory factors for the performance of sparse models. Sparse models can approximate more complex functions (Subsection 3.3.1), exploit redundancy in the data (Subsection 2.3.1), and pruning trainable parameters can be seen equivalent to training of the parameters (Section 2.3).

The abilities to specialize more and to approximate more complex functions follow from sparse networks distributing the learnable parameters to a bigger number of neurons. If a sparse network has the same number of HNPs as a dense network, then it also has more hidden neurons. Because each neuron defines one hyperplane in the input space, then the sparse network defines more hyperplanes than the dense

network with the same number of HNPs. These benefits of sparse architectures apply to all sparse networks, regardless of whether the sparsity is obtained by randomized or optimized pruning.

However, random sparsity is not enough for a sparse architecture to have the two last benefits of sparsity; optimized sparsity is needed to exploit redundancy in the data or to be trained by pruning. Phenomena like the Lottery Ticket Hypothesis have been empirically verified in different problem domains, showing that sparse architectures can excel outside simple image classification tasks. All four benefits from sparsity apply to the winning tickets described in the LTH. However, the relationship of the explanatory factors which contribute to the success of winning tickets is an unstudied question and left for future work.

In Section 6.3, I presented results from experiments with different hyperparameters of MBH[1]. The minimum coverage c_P affects the measured specialization significantly: with $c_P > 0.95$ the large sparse networks did not specialize more than the small dense networks. I found that the specialization of networks develops linearly with all the networks until a threshold value $c_P \approx 0.9$, after which the specialization measure of sparse networks declines quickly and ends up with the dense networks.

I hypothesize that the reasonable value for the minimum coverage stands in relation to the validation accuracy learned by the networks. If the networks do not learn to distinguish between the classes in the data, they cannot be expected to specialize to phenomena of similar abstraction. The comprehensive study on the effects the minimum coverage is left for future work.

This work is the first to compute the number local activation regions created by sparse networks. The experiment results support the view presented in [30, 31] that the expressive power of neural networks does not grow exponentially with network depth, but instead with the number of neurons. This has been predicted by [74] for networks that have less neurons than their input dimension, which is the case with the networks used in these experiments. My experiments contribute to the expressivity conversation by empirically demonstrating that the aforementioned results seem to hold for sparse networks as well.

One interesting finding is that, within the used datasets, specialization correlates positively with the validation accuracy (Figure 6.4), but between the datasets the correlation is negative instead (figures 6.2 and 6.3). The highest validation accuracy Lenets with 9.5k HNPs reached on the Digit MNIST was $\approx 96.5\%$, and on the Fashion MNIST the top accuracy was $\approx 87.5\%$ (Figure 6.2). The same networks reached specialization[2] of 0.4 on Digit MNIST, and 0.6 on Fashion MNIST (Figure 6.3). In

[1] See Section 5.2 for discussion on the hyperparameters.

[2] With minimum coverage $c_P = 0.95$.

other words, heavily pruned networks reached the *higher* specialization on the dataset, where their validation accuracy was *lower*.

The negative correlation between validation accuracy and network specialization between the datasets suggests that networks which learn i) to classify inputs with high accuracy and ii) the underlying input distribution specialize *less* than the networks which reach high accuracy but have overfitted to the data. This is intuitive, since specializing less can be seen as generalizing more, which is associated with networks that learn the underlying input distribution properly.

With this in mind, specialization measure and MBH could potentially be used to measure the overfitting of FFNNs. If a FFNN has a high specialization score then samples from outside the dataset are less likely to activate the specialized subnetwork. The other way around, networks with a high validation accuracy and small specialization can be expected to generalize better to samples from outside the dataset than the ones with large specialization.

There is no theoretical limitation that prevents applying MBH to any FFNN with a finite input space. For example, the specialization of any of the feed forward layers of GPT-3, an autoregressive language model with 175 billion parameters, could be analyzed with the MBH, given a dataset with binary labeling w.r.t. some semantic phenomenon in the data. Applying specialization in different domains and more complex architectures is left for future work due to space and resource limitations of this work.

This work describes a framework to measure how specialized a neural network is to high level features in the data. My hypothesis, that sparse networks specialize more than dense networks with the same number of hidden network parameters, was supported by the experiment results presented in Chapter 6. The domain and architecture agnostic MBH algorithm, which computes the specialization of a FFNN, could potentially be used to evaluate the overfitting of FFNNs in the future work.

Acknowledgements

I thank Antti Ukkonen for introducing me to the Lottery Ticket Hypothesis, and for all the insightful conversations we have had since. I would like to thank everyone who has reviewed and commented drafts of this work, especially Fiona Melzer for proofreading the whole text. In addition, the social distancing measures taken by several nations during the spring and summer of 2020 deserve a special mention: this work would not have been completed without them.

Bibliography

[1] V. Akinwande, C. Cintas, S. Speakman, and S. Sridharan. Identifying Audio Adversarial Examples via Anomalous Pattern Detection. *arXiv:2002.05463 [cs, eess, stat]*, Feb. 2020. arXiv: 2002.05463 version: 1.

[2] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. Network Dissection: Quantifying Interpretability of Deep Visual Representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Apr. 2017. arXiv: 1704.05796.

[3] D. Bau, J.-Y. Zhu, H. Strobelt, B. Zhou, J. B. Tenenbaum, W. T. Freeman, and A. Torralba. GAN Dissection: Visualizing and Understanding Generative Adversarial Networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, page 19, 2019.

[4] Y. Bengio and O. Delalleau. On the Expressive Power of Deep Architectures. In J. Kivinen, C. Szepesvári, and E. Ukkonen, editors, *Algorithmic Learning Theory*, volume 6925, pages 18–36. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. Series Title: Lecture Notes in Computer Science.

[5] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language Models are Few-Shot Learners. *arXiv:2005.14165 [cs]*, June 2020. arXiv: 2005.14165.

[6] C. Bucilua, R. Caruana, and A. Niculescu-Mizil. Model Compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Philadelphia, Pennsylvania, USA, Aug. 2006.

[7] S. Cattell. Geometric Decomposition of Feed Forward Neural Networks. *arXiv:1612.02522 [cs, math]*, Dec. 2016. arXiv: 1612.02522.

[8] C.-H. Cheng, G. Nührenberg, and H. Yasuoka. Runtime Monitoring Neuron Activation Patterns. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 300–303, Florence, Italy, Sept. 2018. arXiv: 1809.06573.

[9] J. Collins, J. Sohl-Dickstein, and D. Sussillo. Capacity and Trainability in Recurrent Neural Networks. In *arXiv:1611.09913 [cs, stat]*, Mar. 2017. arXiv: 1611.09913.

[10] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv:1602.02830 [cs]*, Feb. 2016. arXiv: 1602.02830.

[11] S. Desai, H. Zhan, and A. Aly. Evaluating Lottery Tickets Under Distributional Shifts. In *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo 2019)*, pages 153–162, Hong Kong, Oct. 2019. Association for Computational Linguistics. arXiv: 1910.12708 version: 1.

[12] T. Dettmers and L. Zettlemoyer. Sparse Networks from Scratch: Faster Training without Losing Performance. *arXiv:1907.04840 [cs, stat]*, Aug. 2019. arXiv: 1907.04840.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*, May 2019. arXiv: 1810.04805.

[14] S. Dey, K.-W. Huang, P. A. Beerel, and K. M. Chugg. Pre-Defined Sparse Neural Networks With Hardware Acceleration. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):332–345, June 2019.

[15] R. Ehlers. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *Automated Technology for Verification and Analysis*, volume Automated Technology for Verification and Analysis, May 2017. arXiv: 1705.01320.

[16] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen. Rigging the Lottery: Making All Tickets Winners. *arXiv:1911.11134 [cs, stat]*, Nov. 2019. arXiv: 1911.11134.

[17] U. Evci, F. Pedregosa, A. Gomez, and E. Elsen. The Difficulty of Training Sparse Neural Networks. In *ICML 2019 Workshop Deep Phenomena Blind*, July 2019. arXiv: 1906.10732.

[18] J. Frankle and M. Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *International Conference on Learning Representations*, Mar. 2019. arXiv: 1803.03635.

[19] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin. Stabilizing the Lottery Ticket Hypothesis. *arXiv:1903.01611 [cs, stat]*, June 2019. arXiv: 1903.01611 version: 2.

[20] T. Gale, E. Elsen, and S. Hooker. The State of Sparsity in Deep Neural Networks. *arXiv:1902.09574 [cs, stat]*, Feb. 2019. arXiv: 1902.09574.

[21] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal. Explaining Explanations: An Overview of Interpretability of Machine Learning. In *The 5th IEEE International Conference on Data Science and Advanced Analytics (DSAA 2018). [Research Track]*, Feb. 2019. arXiv: 1806.00069.

[22] X. Glorot, A. Bordes, and Y. Bengio. Deep Sparse Rectier Neural Networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume Volume 15 of JMLR: W&CP 15., page 9, 2011.

[23] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing Deep Convolutional Networks using Vector Quantization. *arXiv:1412.6115 [cs]*, Dec. 2014. arXiv: 1412.6115.

[24] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[25] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. In *Advances in neural information processing systems*, pages 2672–2680, June 2014. arXiv: 1406.2661.

[26] D. Gopinath, H. Converse, C. S. Pasareanu, and A. Taly. Property Inference for Deep Neural Networks. In *The 34th IEEE/ACM International Conference on Automated Software Engineering*, San Diego, CA, USA, Apr. 2019. arXiv: 1904.13215.

[27] R. H. R. Hahnloser and H. S. Seung. Permitted and Forbidden Sets in Symmetric Threshold-Linear Networks. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 217–223. MIT Press, 2001.

[28] S. Han, H. Mao, and W. J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv:1510.00149 [cs]*, Feb. 2016. arXiv: 1510.00149.

[29] S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both Weights and Connections for Efficient Neural Networks. *Advances in neural information processing systems*, pages 1135–1143, Oct. 2015. arXiv: 1506.02626.

[30] B. Hanin and D. Rolnick. Complexity of Linear Regions in Deep Networks. In *arXiv:1901.09021 [cs, math, stat]*, June 2019. arXiv: 1901.09021.

[31] B. Hanin and D. Rolnick. Deep ReLU Networks Have Surprisingly Few Activation Patterns. In *NeurIPS 2019*, June 2019. arXiv: 1906.00904.

[32] G. Hinton, S. Sabour, and N. Frosst. Matrix capsules with EM routing. page 15, 2018.

[33] G. Hinton, O. Vinyals, and J. Dean. Distilling the Knowledge in a Neural Network. *arXiv:1503.02531 [cs, stat]*, Mar. 2015. arXiv: 1503.02531.

[34] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580 [cs]*, July 2012. arXiv: 1207.0580.

[35] F. Hohman, H. Park, C. Robinson, and D. H. Chau. Summit: Scaling Deep Learning Interpretability by Visualizing Activation and Attribution Summarizations. Sept. 2019. arXiv: 1904.02323.

[36] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, Jan. 1989.

[37] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [cs]*, Apr. 2017. arXiv: 1704.04861.

[38] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang. Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures. *arXiv:1607.03250 [cs]*, July 2016. arXiv: 1607.03250.

[39] G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, Honolulu, HI, July 2017. IEEE.

[40] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. *The Journal of Machine Learning Research*, pages 6869–6998, 2017.

[41] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. Chau. ActiVis: Visual Exploration of Industry-Scale Deep Neural Network Models. In *arXiv:1704.01942 [cs, stat]*, pages 88–97, Aug. 2017. arXiv: 1704.01942.

[42] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling Laws for Neural Language Models. *arXiv:2001.08361 [cs, stat]*, Jan. 2020. arXiv: 2001.08361.

[43] R. M. Karp. Reducibility among Combinatorial Problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, The IBM Research Symposia Series, pages 85–103. Springer US, Boston, MA, 1972.

[44] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. Feb. 2017. arXiv: 1702.01135.

[45] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, and R. Sayres. Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV). In *arXiv:1711.11279 [stat]*, June 2018. arXiv: 1711.11279.

[46] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[47] A. Kumar, T. Serra, and S. Ramalingam. Equivalent and Approximate Transformations of Deep Neural Networks. *arXiv:1905.11428 [cs, stat]*, May 2019. arXiv: 1905.11428.

[48] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[49] Y. LeCun, L. Bottou, Y. Bengio, and P. Ha. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324:46, Nov. 1998.

[50] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal Brain Damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan-Kaufmann, 1990.

[51] N. Lee, T. Ajanthan, and P. H. S. Torr. SNIP: Single-shot Network Pruning based on Connection Sensitivity. Feb. 2019. arXiv: 1810.02340.

[52] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the Value of Network Pruning. *arXiv:1810.05270 [cs, stat]*, Mar. 2019. arXiv: 1810.05270.

[53] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang. The Expressive Power of Neural Networks: A View from the Width. In *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA., Nov. 2017. arXiv: 1709.02540.

[54] X. Ma, F.-M. Guo, W. Niu, X. Lin, J. Tang, K. Ma, B. Ren, and Y. Wang. PCONV: The Missing but Desirable Sparsity in DNN Weight Pruning for Real-time Execution on Mobile Devices. In *arXiv:1909.05073 [cs, stat]*, Jan. 2020. arXiv: 1909.05073.

[55] E. Malach, G. Yehudai, S. Shalev-Shwartz, and O. Shamir. Proving the Lottery Ticket Hypothesis: Pruning is All You Need. *arXiv:2002.00585 [cs, stat]*, Feb. 2020. arXiv: 2002.00585 version: 1.

[56] L. Meegahapola, V. Subramaniam, L. Kaplan, and A. Misra. Prior Activation Distribution (PAD): A Versatile Representation to Utilize DNN Hidden Units. *arXiv:1907.02711 [cs]*, July 2019. arXiv: 1907.02711 version: 1.

[57] R. Mehta. Sparse Transfer Learning via Winning Lottery Tickets. In *arXiv:1905.07785 [cs, stat]*, Dec. 2019. arXiv: 1905.07785 version: 2.

[58] G. Melis, C. Dyer, and P. Blunsom. On the State of the Art of Evaluation in Neural Language Models. *arXiv:1707.05589 [cs]*, Nov. 2017. arXiv: 1707.05589.

[59] G. Montufar. Notes on the number of linear regions of deep neural networks. Tallinn, Estonia, Mar. 2017.

[60] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio. On the Number of Linear Regions of Deep Neural Networks. In Z. Ghahramani, M. Welling, C. Cortes,

N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2924–2932. Curran Associates, Inc., 2014.

[61] A. S. Morcos, H. Yu, M. Paganini, and Y. Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In *arXiv:1906.02773 [cs, stat]*, Oct. 2019. arXiv: 1906.02773.

[62] R. Novak, Y. Bahri, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein. Sensitivity and Generalization in Neural Networks: an Empirical Study. *arXiv:1802.08760 [cs, stat]*, June 2018. arXiv: 1802.08760.

[63] R. Pascanu, G. Montufar, and Y. Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv:1312.6098 [cs]*, Feb. 2014. arXiv: 1312.6098.

[64] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8026–8037. Curran Associates, Inc., 2019.

[65] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. S. Dickstein. On the Expressive Power of Deep Neural Networks. In *Proceedings of the 34 th International Conference on Machine Learning*, page 8, Sydney, Australia, 2017.

[66] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein. Survey of Expressivity in Deep Neural Networks. *arXiv:1611.08083 [cs, stat]*, Nov. 2016. arXiv: 1611.08083.

[67] C. E. Rasmussen and Z. Ghahramani. Occam's Razor. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 294–300. MIT Press, 2001.

[68] A. Renda, J. Frankle, and M. Carbin. Comparing Rewinding and Fine-tuning in Neural Network Pruning. In *arXiv:2003.02389 [cs, stat]*, Mar. 2020. arXiv: 2003.02389.

[69] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International journal of computer vision*, 115(3):211–252, Jan. 2015. arXiv: 1409.0575.

[70] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic Routing Between Capsules. In *arXiv:1710.09829 [cs]*, Oct. 2017. arXiv: 1710.09829.

[71] P. Savarese, H. Silva, and M. Maire. Winning the Lottery with Continuous Sparsification. *arXiv:1912.04427 [cs, stat]*, Dec. 2019. arXiv: 1912.04427 version: 1.

[72] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni. Green AI. *arXiv:1907.10597 [cs, stat]*, Aug. 2019. arXiv: 1907.10597.

[73] T. Serra, A. Kumar, and S. Ramalingam. Lossless Compression of Deep Neural Networks. *arXiv:2001.00218 [cs, math, stat]*, Jan. 2020. arXiv: 2001.00218.

[74] T. Serra, C. Tjandraatmadja, and S. Ramalingam. Bounding and Counting Linear Regions of Deep Neural Networks. In *arXiv:1711.02114 [cs, math, stat]*, Nov. 2017. arXiv: 1711.02114.

[75] R. P. Stanley. An Introduction to Hyperplane Arrangements. *Lecture Notes*, (Lect. notes, IAS/Park City Math. Inst):114, 2016.

[76] E. Strubell, A. Ganesh, and A. McCallum. Energy and Policy Considerations for Deep Learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy, July 2019. Association for Computational Linguistics.

[77] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *arXiv:1602.07261 [cs]*, Aug. 2016. arXiv: 1602.07261.

[78] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv:1312.6199 [cs]*, Dec. 2013. arXiv: 1312.6199.

[79] Y. Tian. Student Specialization in Deep ReLU Networks With Finite Width and Input Dimension. *arXiv:1909.13458 [cs, stat]*, Nov. 2019. arXiv: 1909.13458.

[80] Y. Tian, T. Jiang, Q. Gong, and A. Morcos. Luck Matters: Understanding Training Dynamics of Deep ReLU Networks. *arXiv:1905.13405 [cs, stat]*, June 2019. arXiv: 1905.13405.

[81] C. Urban, M. Christakis, V. Wüstholz, and F. Zhang. Perfectly Parallel Fairness Certification of Neural Networks. *arXiv:1912.02499 [cs]*, Apr. 2020. arXiv: 1912.02499.

[82] C. Wang, G. Zhang, and R. Grosse. Picking Winning Tickets Before Training by Preserving Gradient Flow. In *arXiv:2002.07376 [cs, stat]*, Feb. 2020. arXiv: 2002.07376.

[83] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning Structured Sparsity in Deep Neural Networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2074–2082. Curran Associates, Inc., 2016.

[84] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated Residual Transformations for Deep Neural Networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, Apr. 2017. arXiv: 1611.05431.

[85] T.-J. Yang, Y.-H. Chen, and V. Sze. Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Apr. 2017. arXiv: 1611.05128.

[86] H. Yu, S. Edunov, Y. Tian, and A. S. Morcos. Playing the lottery with rewards and multiple languages: lottery tickets in RL and NLP. *arXiv:1906.02768 [cs, stat]*, Nov. 2019. arXiv: 1906.02768 version: 2.

[87] T. Zaslavsky. Counting the faces of cut-up spaces. *Bulletin of The American Mathematical Society - BULL AMER MATH SOC*, 81, Sept. 1975.

[88] Q.-s. Zhang and S.-c. Zhu. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, 19(1):27–39, Jan. 2018.

[89] S. Zhang, Y. Wu, T. Che, Z. Lin, R. Memisevic, R. R. Salakhutdinov, and Y. Bengio. Architectural Complexity Measures of Recurrent Neural Networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1822–1830. Curran Associates, Inc., 2016.

[90] H. Zhou, J. Lan, R. Liu, and J. Yosinski. Deconstructing Lottery Tickets: Zeros, Signs, and the Supermask. In *arXiv:1905.01067 [cs, stat]*, Vancouver, Canada, Sept. 2019. arXiv: 1905.01067.

[91] M. Zhu and S. Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv:1710.01878 [cs, stat]*, Nov. 2017. arXiv: 1710.01878.

Appendix A.

A.1 Network Hyperparameters

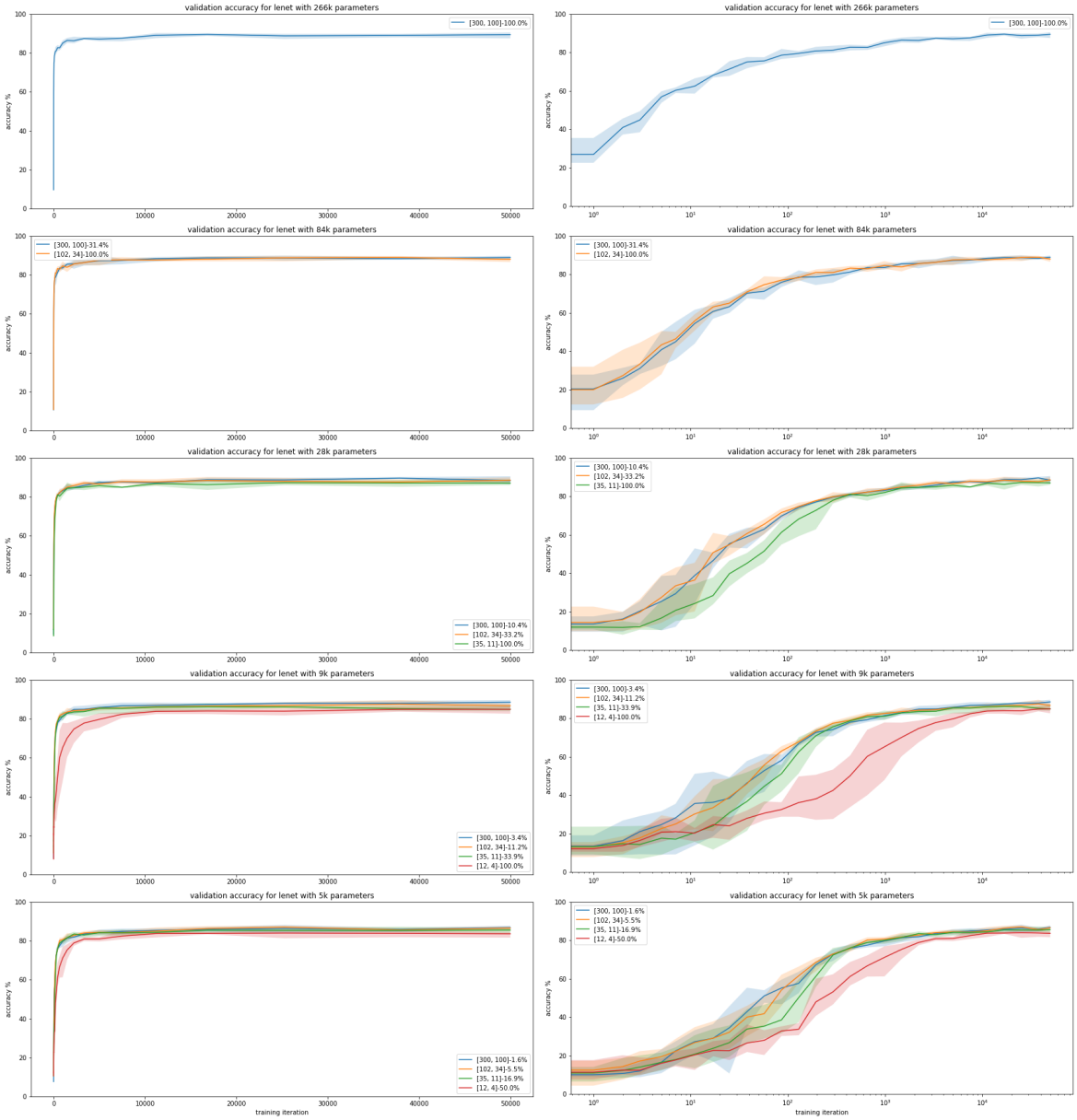
The hyperparameters of large, dense Lenets are the same as in [18], except that the weights are initialized with initialization scheme presented in [31].

Hyperparameter	Large dense	Large sparse	Small dense
Neurons	410	410	26
Parameters	266610	10482	9522
Density (with output l.)	100%	3.9%	100%
Hidden neurons	400	400	16
HNPs	265600	9472	9472
Density	100%	3.6%	100%

Training hyperparameter	Shared by all the networks
Batch size	60
Training iterations	50k
Bias std	10^{-6}
Optimizer	Adam
Learning rate	$1.2e-3$
Loss function	Cross Entropy Loss

A.2 Validation Accuracy of Lenets Over Training

Since all the models are trained for constant 50k iterations, some of then might not converge, or some might overfit. However, this is not the case. In Figure A.1 the validation accuracy of Lenet models trained on Fashion MNIST is plotted over the training, linear and logarithmic scale.



(a) **(b)**

Figure A.1: Validation accuracies from training of the Lenet FFNNs used in the experiments. Dataset is Fashion MNIST. (a) and (b) have the same data, (a) has linear and (b) has logarithmic x-axis. None of the models overfit.

A.3 The Effect of Data Normalization

I experimented, how robust specialization measured with the MBH is against transformations performed on the data. I normalized every dimension independently to have the mean 0 and std 1, and run the experiments described in sections 6.2 and 6.3 on models created with the same hyperparameter setups as the ones used to experiment on the regular Fashion MNIST.

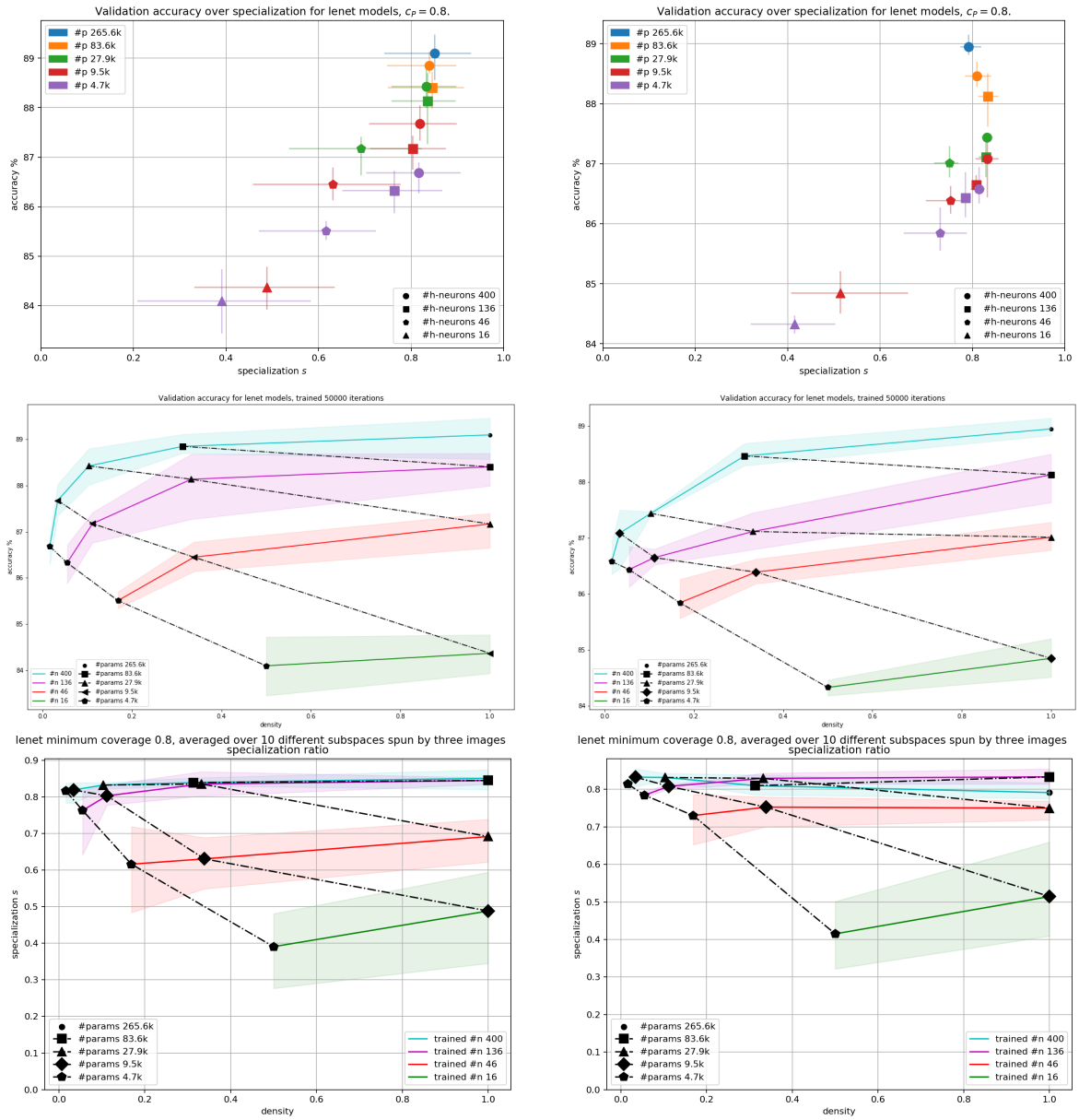
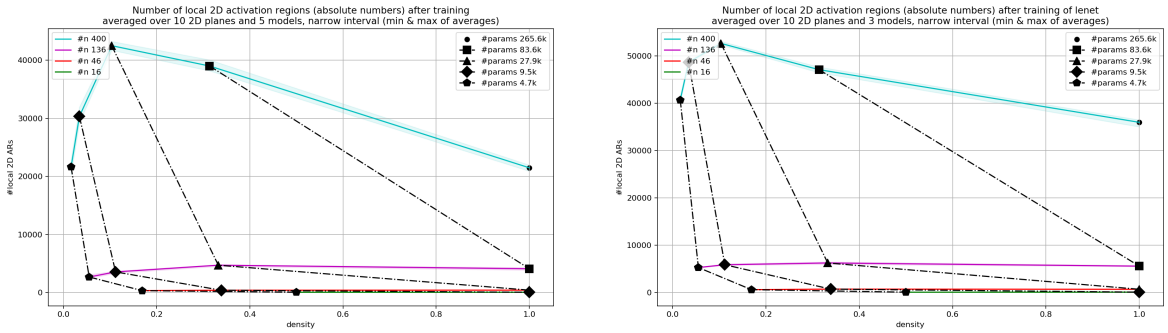


Figure A.2: Accuracy over specialization (row 1), accuracy over densities (row 2), and specialization (row 4) for regular (column 1) and normalized (column 2) Fashion MNIST. The hyperparameter setups are exactly the same, although regular has 5, and normalized 3, models per setup. The y-scales may differ between the columns.

I found that the trend in specialization was the same for models trained on the normalized data, as seen in Figure A.2. The validation accuracy of pruned models was weaker with the transformed data. In addition, the smaller, more dense models specialized more on the normalized data, although they did not break the trend of larger, more sparse models specializing more. The number of activation regions, however, was clearly affected by the normalization: large sparse models produced roughly 20% more activation regions, when they were trained on normalized data. This can be seen in Figure A.3.

The larger number of activation regions with normalized data is expected, because normalization moves data around the origin, and all hyperplanes are tightly around the origin at the initialization. Because neurons biases are initialized small, i.e. mean 0 and standard deviation 10^{-6}, all of them are near the origin. Since all hyperplanes are in the middle of the data at the initialization, it is not surprising that the 2D planes spun by random datapoints from the validation dataset are split by more neurons, and therefore split to more regions.



(a) **(b)**

Figure A.3: Absolute number of 2D ARs over densities for regular (column 1) and normalized (column 2) Fashion MNIST. The hyperparameter setups are exactly the same, although regular has 5, and normalized 3, models per setup. The y-scales are different between the columns.

Appendix B.

B.1 Minimal Blanket

It turns out that the minimal blanket is the intersection of all the blankets in $\mathbf{B}$, when the $c_P = 1$.

Lemma B.1. *Minimal blanket is the intersection of blankets, when* $c_P = 1$.

Proof. Let F be a neural network, and $\mathbf{S}$ be a set of its specialized subnetworks w.r.t. a semantic phenomenon P. The minimal blanket B_0 of P is the intersection I_B of blankets that are defined by the subnetworks in $\mathbf{S}$.

It is sufficient to show that 1) the intersection I_B of blankets in $\mathbf{B}$ covers the phenomenon P, i.e. it is a blanket of P, and that 2) the size of I_B is smaller than or equal to the size of individual blankets in $\mathbf{B}$.

1. Since all the subnetworks in $\mathbf{S}$ are specialized to P, all the blankets in $\mathbf{B}$ cover the part of the input space which embodies P. Therefore the intersection I of the blankets in $\mathbf{B}$ also covers the part of the input space which embodies P, and it is a blanket of P.
2. Because the blankets are unions of activation regions, then I is also a union of activation regions. Since I is the intersection of the blankets in $\mathbf{B}$, it cannot contain any regions that would not be in the smallest blanket in $\mathbf{B}$, and therefore it cannot be larger than the smallest blanket in $\mathbf{B}$.

□

When $c_P < 1$, the greedy algorithm to find the minimal blanket (Appendix C.1) ensures that blankets with smaller minimum coverage are covered by blankets, with a larger minimum coverage. This can be seen in figures B.1 and B.2, where a small dense and large sparse Lenet define minimal blankets over all 10 classes present in the Fashion MNIST dataset.

The plane, on which the minimal blanket coverage is measured, is spun by three images: classes 0, 1, and 9, one from each. This results the blankets of these classes to concentrate around the image, which is from that class. It is interesting to note, how minimal blankets of the other 7 classes behave on this plane: some classes, for example 6 and 8, have rather large blankets with both models. On the contrary, the minimal blanket of class 7 does not collide with the plane at all with the large model and $c_P < 1$ (Figure B.2).

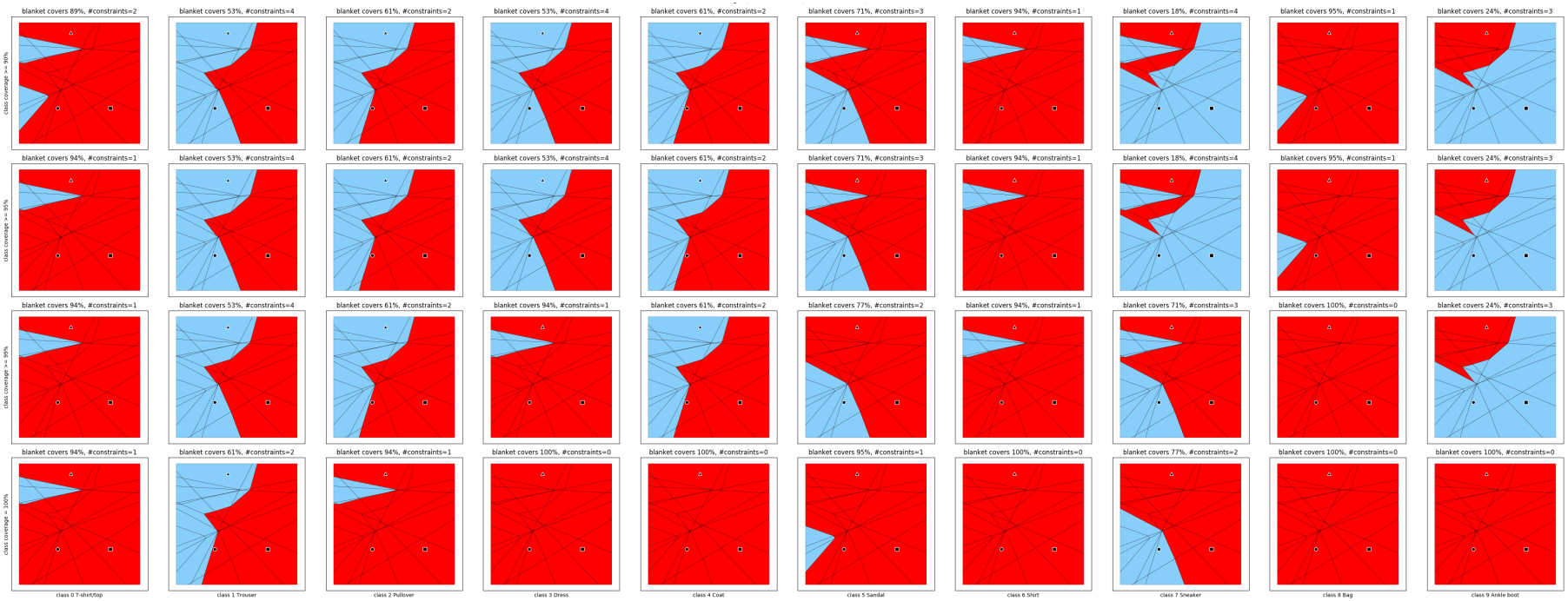


Figure B.1: A small dense Lenet has specialized subnetworks to cover the ten semantic phenomena (columns) labeled in Fashion MNIST -dataset as classes with four different minimum coverage c_P (rows). The three images spanning the 2D plane are from classes 0, 1 and 9. The network has (12,4,10) neurons, density 1, and 9472 HNPs.

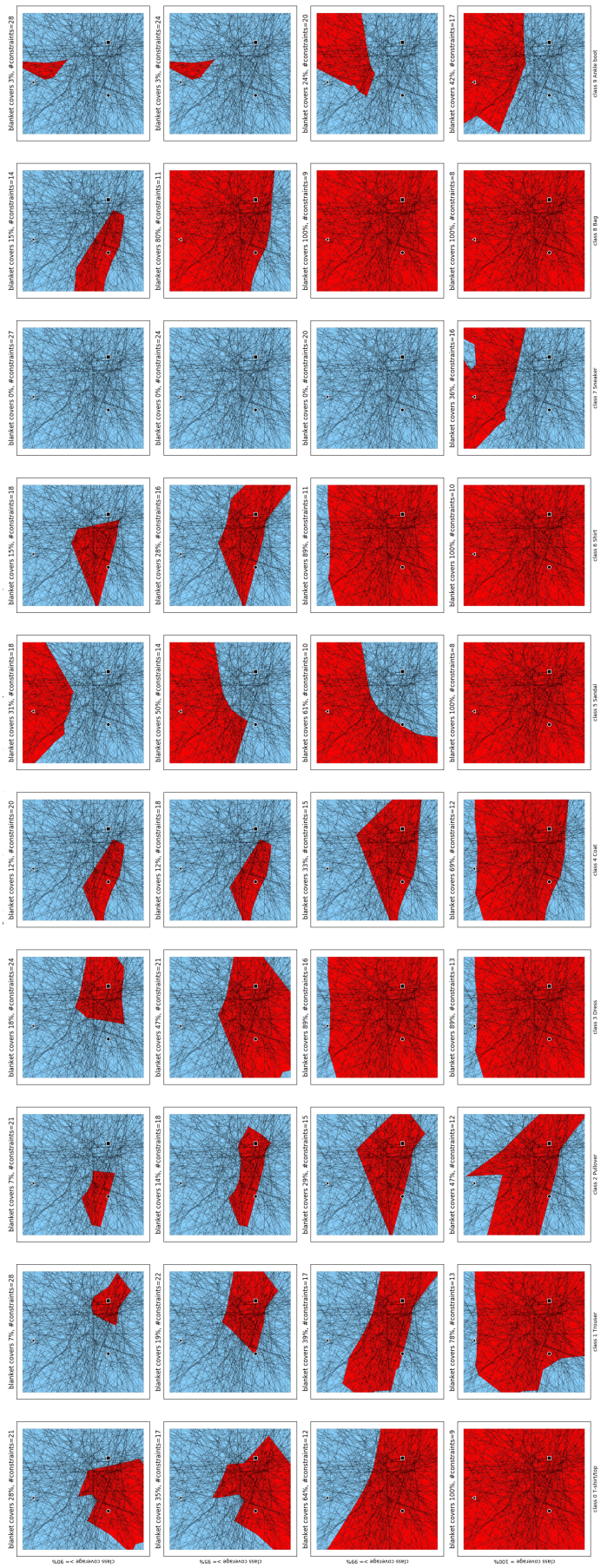


Figure B.2: A large sparse Lenet has specialized subnetworks to cover the ten semantic phenomena (columns) labeled in Fashion MNIST -dataset as classes with four different minimum coverage c_P (rows). The three images spanning the 2D plane are from classes 0, 1 and 9. The network has (300,100,10) neurons, density 0.036, and 9472 HNPs.

Appendix C.

C.1 Greedy Algorithm to Find the Maximal Pattern

The algorithm is guaranteed to make the blanket smaller on every step because intersections cannot be larger than the constituents (see Lemma 5.1). However, the greedy algorithm is not guaranteed to find the minimal blanket when the minimum coverage $c_P < 1$. A scenario, where the greedy algorithm fails to find the actual minimal blanket, is visualized in Figure C.1.

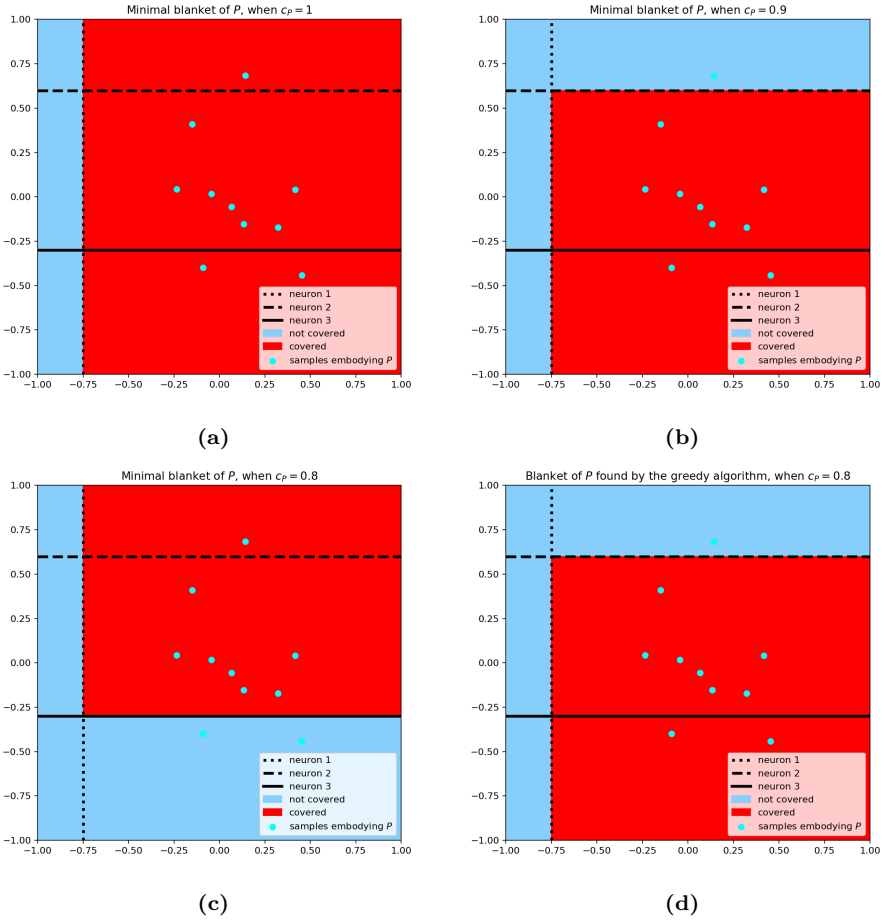


Figure C.1: Illustration how the greedy algorithm can fail to find the minimal blanket when $c_P < 1$. Three neurons split the finite input space $\mathbb{R}_{in}$ into 6 regions, $\mathbb{R}_{in} = \{\ x \mid x_1, x_2 \in [-1, 1],\ x \in \mathbb{R}^2\}$. There are 10 samples that embody some semantic phenomenon P in the input space. In the subfigures a-c there is the actual minimal blanket, when the minimum coverage c_P is 1, 0.9, and 0.8, respectively. In (d) there is presented the blanket the greedy algorithm finds, when $c_P = 0.8$. Note that the minimal blanket for $c_P = 0.8$ is the one in (c).

Appendix D.

D.1 Mining Decision Patterns with Decision Tree Learning

Activation and decision patterns are argued to hold a lot information about the decision process of the neural network [26, 56]. However, the research on what type of information and how the information could be accessed is still ongoing.

To evaluate, how much information decision patterns have about the semantic properties of the inputs, I applied the mining of decision patterns using decision trees (DTs) [26] as a part of the research I did for this work. Described briefly, the experiment consisted of

1. processing validation dataset with neural networks, recording the layer patterns of the last hidden layer,
2. training decision trees using the labeled layer patterns from 1) to classify the inputs, and
3. measuring properties of the decision trees.

The DTs were not pruned, i.e., each of the leaves is pure w.r.t. the class labels in the data. I performed the experiment on models prior and post training.

The properties measured were the depth of the tree, the number of leaves, and the average leaf depth. Because all the statistics were really stable w.r.t. the DT, I trained only 1 tree per network and scenario (at initialization / trained).

The results of the Lenets trained on Fashion MNIST can be seen in Figure D.1. The DT can learn to classify inputs with 100% accuracy by observing only the the last hidden layer's layer pattern both before and after training. This is expected, since the DT is allowed to overfit (each leaf is pure, and the depth is not restricted). In addition, the tree depth average leaf depth were really stable, regardless of the scenario.

However, the number of leaves in the decision tree got significantly smaller as the result of the training (Figure D.1, the second plot). In practice this means that the number of *unique decision patterns* that is needed to classify the 10k validation images of Fashion MNIST with 100% accuracy, gets significantly smaller by training the network. In other words, training the networks concentrates the semantically similar inputs to support a smaller number of unique decision patterns.

Another interesting observation about the unique patterns needed to classify the validation set, is that *fewer* decision patterns hold all relevant information, when

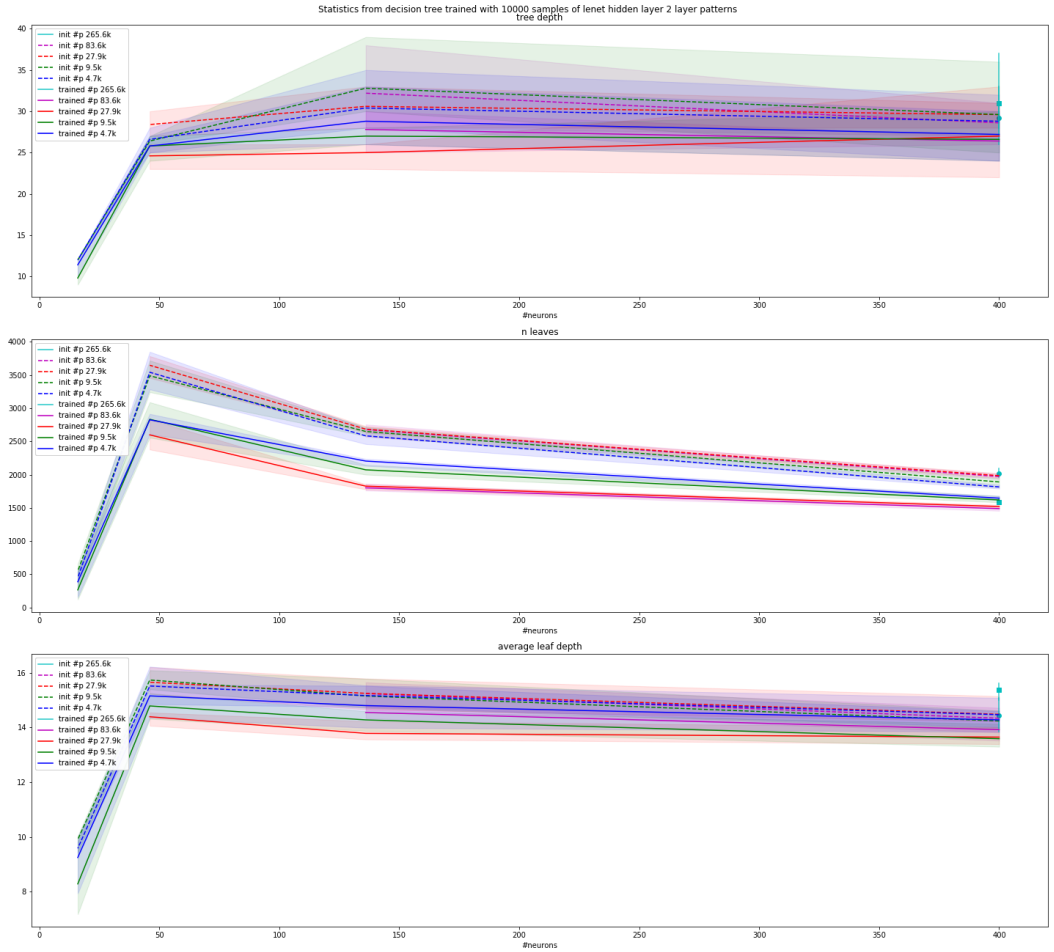


Figure D.1: Statistics computed from decision trees, which were trained on the layer patterns of the last hidden layers of Lenet models. Dashed lines are DTs trained with patterns from un-trained neural networks, solid lines are with patterns from the same networks after training. The first plot is the depth of the tree, the second is the number of leaves in the tree, and the third the average depth of a leaf. All three statistics are plotted over the number of hidden neurons in the network.

the layer pattern grows *larger*, i.e., there are more potential patterns to choose from. Moving from a network with (12,4) hidden neurons to network with (35,11) hidden neurons, the number of unique patterns (leaves of the DT) grows. After that, however, increasing the number of neurons in the last hidden layer up to 100 does not increase the number of unique patterns, but instead the number of leaves grows smaller. This suggests that after reaching a critical number of hidden neurons the network continues to concentrate the semantic phenomena to smaller number of subnetworks. It is good to note that the number of decision patterns (subnetworks) mined with DT learning is still an order of magnitude larger, than the number of classes the network and the DT are required to classify the data into.

The number of parameters in the networks does not seem to affect, how many unique decision patterns the DT requires to classify the samples, while the number of

neurons in the network correlates with all the metrics.

The number of unique decision patterns mined with the DT learning is broken down to 10 different classes in figures D.2 and D.3. Most of the classes show similar difference with the number of unique patterns between trained and non-trained scenarios, but the absolute number of patterns varies greatly between the classes. For example, the class 1 requires less than 250 unique patterns with all hyperparameter setups before and after training, while class 6 needs almost 700 on networks prior to the training.

I hypothesize that this correlates with the separability of the classes. Class 1 is trousers, which are the only images in the datasets to have vertical black space in the middle (space between the pant legs), while class 6 are shirts that are similar to three other classes (0: T-Shirt/Top, 2: Pullover, and 4: Coat). Besides being similar to other classes, the shirts have a big variety of prints and colors (different shades of gray), which distributes samples from class 6 around the input space. However, the evaluation of this hypothesis is left for future work.

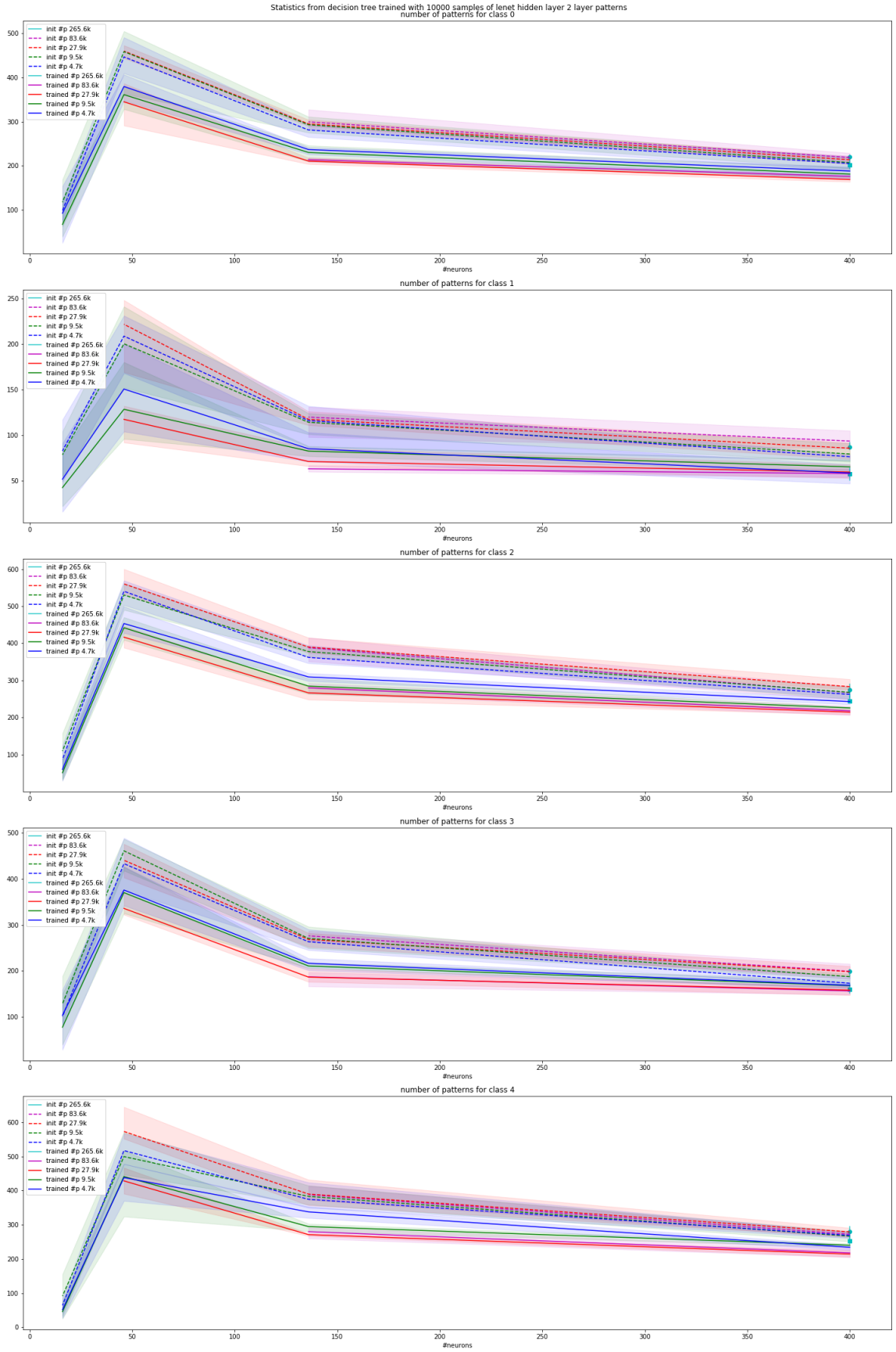


Figure D.2: The number of unique decision patterns on the last hidden layer of Lenet a decision tree requires to classify 100% of the validation dataset (10k samples) of Fashion MNIST. Classes 0-4.

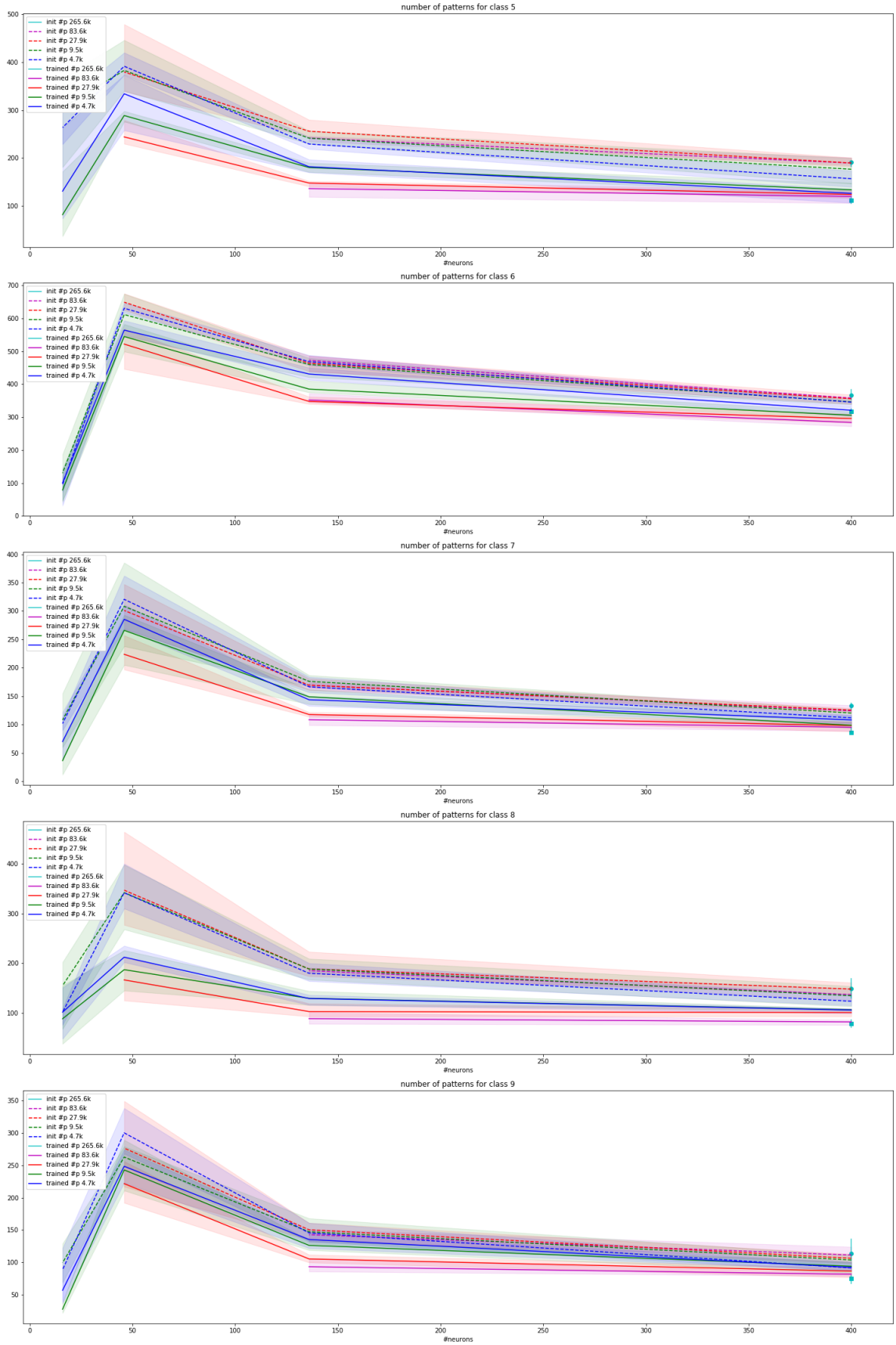


Figure D.3: The number of unique decision patterns on the last hidden layer of Lenet a decision tree requires to classify 100% of the validation dataset (10k samples) of Fashion MNIST. Classes 5-9, continuation to Figure D.2.

Appendix E.

E.1 Unique Activation and Layer Patterns

Recording activation and layer patterns of the network is simple: process an input with the network, record and save which neurons were active. Because the number of activation regions a neural network creates in its input space is large (see Theorem 3.2), each input is probable to produce a unique pattern on the network level (activation patterns, Figure E.1) and on each individual layer (layer patterns, Figure E.2).

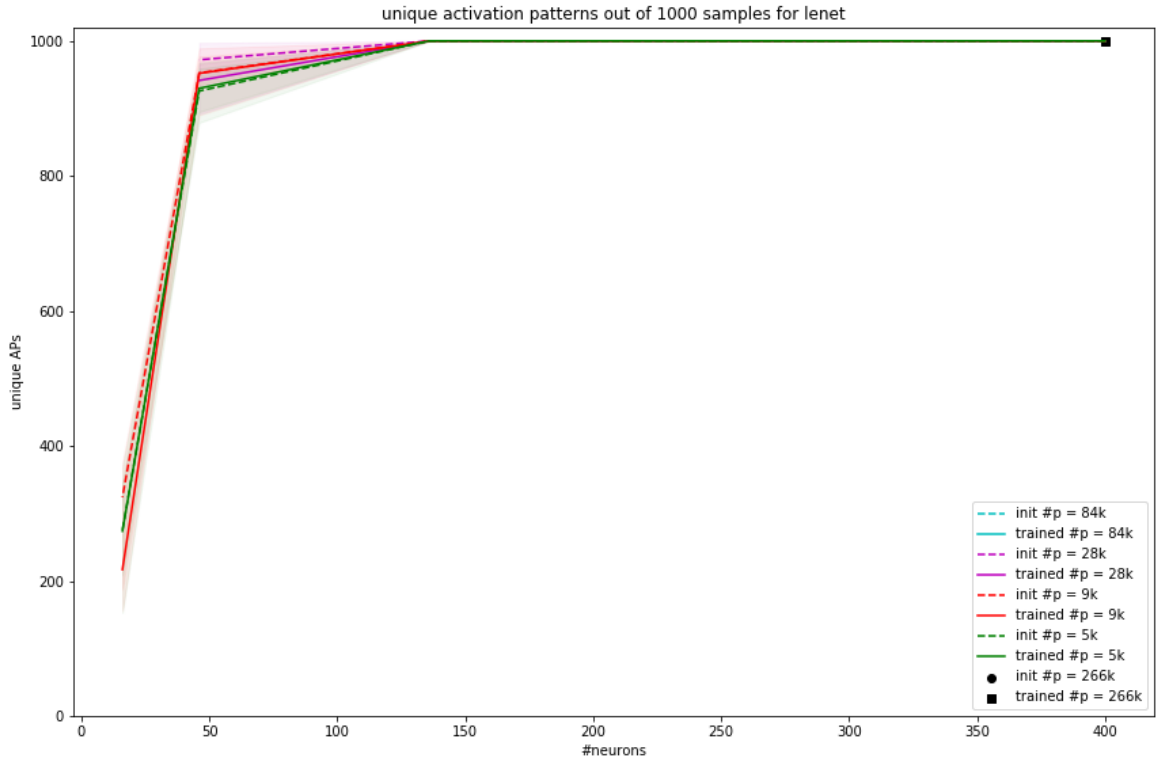
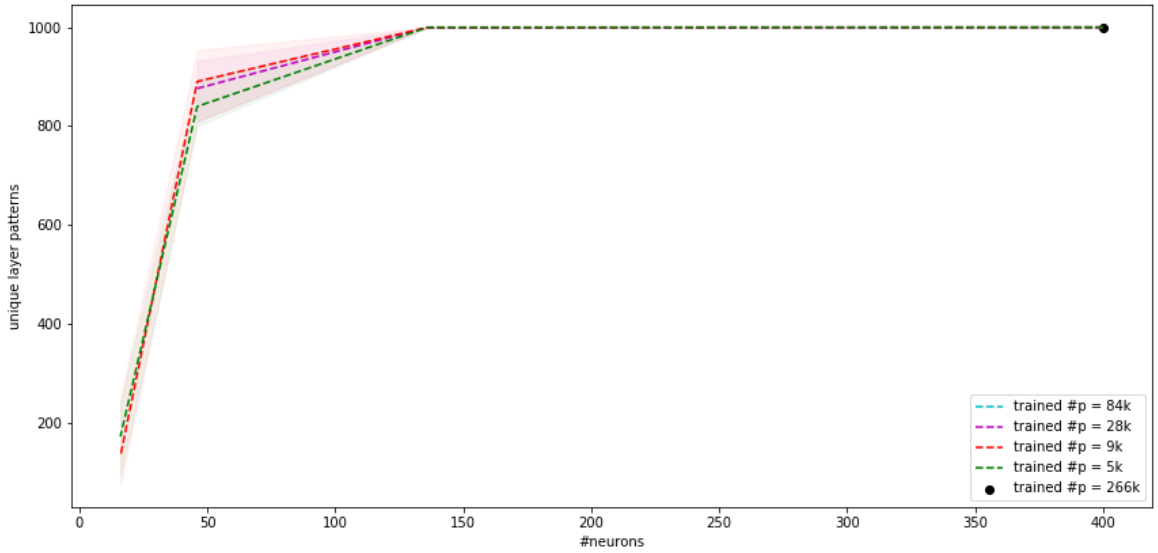


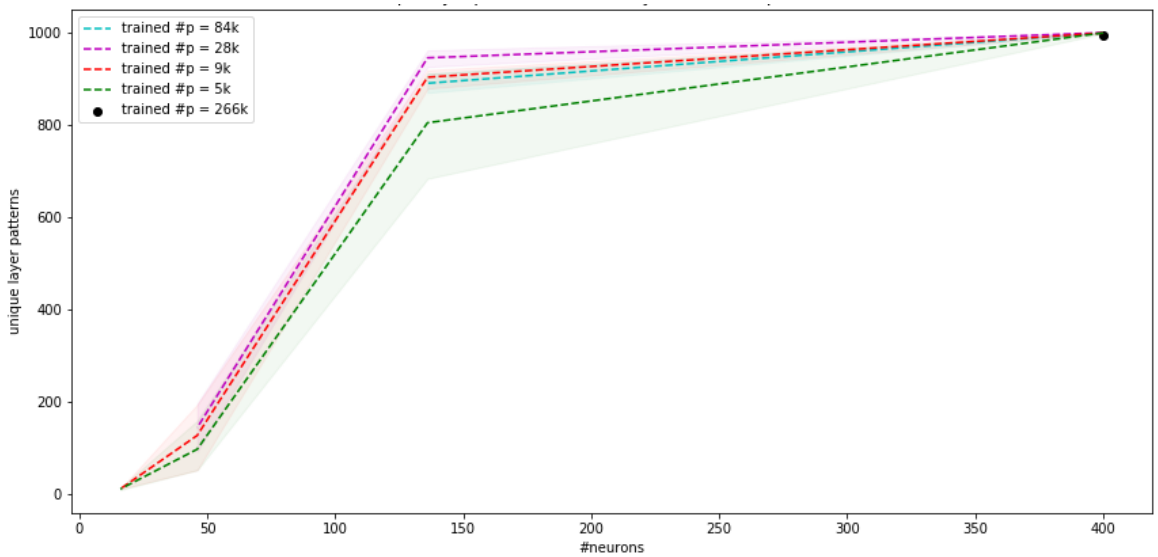
Figure E.1: The number of unique activation patterns of Lenets with different number of HNPs. The x-axis is the number of hidden neurons in the whole network, i.e., the number of neurons considered for the activation pattern (the output layer is excluded). When the network processed 1k images from the validation set, networks with 136 or more hidden neurons produced a unique activation pattern for every single neuron, regardless if the network had been trained or not.

Because each, however how semantically similar, input tends to produce a unique activation pattern when processed by the network, activation patterns are too coarse as a whole to be used to analyze neural networks' decision process. However, observing patterns on the layer level offers some insight even with a limited amount of input samples.

In Figure E.2, the number of unique layer patterns on the last hidden layer of the network is considerably different between networks with 136 hidden neurons and different number of parameters. This suggests that the most pruned networks (5k HNPs) concentrate the processing to a smaller number of neurons, than the networks with the same number of neurons but more parameters.



(a)



(b)

Figure E.2: The number of unique layer patterns of (a) layer 1 (b) layer 2 of Lenets with different number of HNPs. The x-axis is the number of hidden neurons in the whole network, i.e., the number of neurons considered for the activation pattern (the output layer is excluded).